Dissertation zur Erlangung des Doktorgrades
Dr. rer. nat.

Composition and Declassification in Possibilistic Information Flow Security

Thomas Bauereiß

Universität Bremen

Datum des Kolloquiums: 12. September 2019
Erster Gutachter: Prof. Dr. Dieter Hutter
Zweiter Gutachter: Prof. Dr. Bernhard Beckert

Acknowledgments

There are many people without whom this thesis would not have been possible.
I would like to thank my advisor, Dieter Hutter, for giving me the opportunity to work on this thesis and supporting me all the way through.
I thank Bernhard Beckert for taking the time to review my thesis. I also thank the members of my examination committee for their feedback on the thesis and the interesting discussions at the colloquium.
I thank my collaborators for many inspiring discussions, in particular Andrei Popescu. I very much enjoyed working together.
I benefited greatly from the DFG priority programme "Reliably Secure Software Systems" (RS^3). I thank the initiators of RS^3, and in particular the organizer Heiko Mantel and his team, for bringing together so many talented people and fostering collaboration between them.
I thank my coworkers over the years for creating a welcoming, productive, and enjoyable environment, in particular Martin Ring, Helmar Hutschenreuter, Fritjof Bornebusch, Oliver Bösche, and Christian Liguda.
I also thank the administrative staff for keeping everything running smoothly, in particular Lisa Jungmann, Kristiane Schmitt, and Uwe Forgber.
Finally, I thank my friends and my family for their invaluable support.

Contents

Kurzzusammenfassung

Formale Methoden stellen ein Werkzeug dar, um die Abwesenheit ganzer Klassen von Sicherheitslücken nachzuweisen. Obwohl die Forschung in diesem Bereich in den letzten Jahrzehnten große Fortschritte gemacht hat, besteht weiterhin Bedarf an neuen Techniken zur Verbesserung der Skalierbarkeit, z.B. durch Dekomposition der Verifikation nach Systemkomponenten oder Abstraktionsebenen. Erschwerend kommt hinzu, dass auch die geforderten Sicherheitseigenschaften selbst oft komplex sind. Beispielsweise geht es oft nicht darum, bestimmte Informationen unbedingt geheimzuhalten, sondern es gibt detaillierte Anforderungen, wer wann welche Informationen erfahren darf.

In der vorliegenden Dissertation wird ein Ansatz zur Spezifikation und Verifikation von Informationsflusseigenschaften vorgestellt, der auf ausgewählten Arbeiten aus der Literatur aufbaut und deren Vorteile bezüglich Ausdrucksmächtigkeit und Skalierbarkeit vereint. Ausgangsbasis ist ein existierender Ansatz namens Bounded Deducibility (BD) Security, der im Vergleich mit anderen Ansätzen aus der Literatur sehr flexible Informationsflusspolitiken unterstützt, allerdings nur eine Verifikationsmethode mit begrenzter Skalierbarkeit bietet. Das zentrale technische Ergebnis dieser Dissertation ist die Erweiterung von BD Security um ein Kompositionalitätsresultat, das es erlaubt, die Verifikation von Informationsflusseigenschaften eines Systems auf die Verifikation lokaler Eigenschaften einzelner Komponenten des Systems herunterzubrechen. Es gibt gewissen Spielraum bei der Aufteilung der Beweisverpflichtungen auf die einzelnen Komponenten, aber die Sicherheitsgarantien der Komponenten müssen stark genug sein um zueinander zu passen und zusammen die gewünschte Sicherheitseigenschaft des Gesamtsystems zu implizieren. Hinreichende Bedingungen dafür werden in Kapitel 3 formal definiert. Ein Teil dieser Bedingungen ist dabei denen von Mantel's Modular Assembly Kit for Security (MAKS) nachempfunden, während andere neuartig sind und der Ausdrucksmächtigkeit von BD Security bezüglich Informationsflusspolitiken geschuldet sind.

Als Fallstudie wird in Kapitel 4 die kompositionale Verifikation dynamischer Vertraulichkeitseigenschaften in einer Social-Media-Plattform beschrieben, die als verteiltes System auf einer beliebigen Anzahl an Servern aufgesetzt werden kann. In Kapitel 5 wird diskutiert, wie ein bestimmter Verfeinerungsbegriff in einem schrittweisen Entwicklungsprozesses für Workflow-Systeme als Komposition verstanden und mit obigem formalen Rahmenwerk behandelt werden kann, um Sicherheitseigenschaften auf verschiedenen Verfeinerungsebenen zu komponieren. In Kapitel 6 schließlich wird ein Weg diskutiert, um Informationsfluss und andere Arten von Eigenschaften separat zu verifizieren und die Ergebnisse dann mit Hilfe des obigen Kompositionalitätsresultats zu kombinieren.

Chapter 1.

Introduction

1.1. Motivation

Defending the security of a computer system is much harder than attacking it. For an attacker, it is sufficient to find *one* exploitable security hole, while a defender has to consider *all* possible attack vectors. This is what makes formal methods for security so appealing. They allow us to *prove* that whole classes of attacks are mathematically impossible. This can significantly increase the confidence in the security of a system, although it cannot guarantee perfect security: The mathematical models that verification techniques are based on are necessarily abstractions of the concrete systems, and they focus on specific security properties and attacker models. While early models were very coarse-grained, the situation has become much better over the last few decades, with research on formal approaches to security continuously pushing forward the boundaries of what these techniques can achieve. This has led to some impressive results, e.g. the formal verification of functional and security properties of the seL4 operating system kernel, down to the binary level [KAE+14]. This verification has been fully mechanized: all proofs are machine-checked, giving very high confidence in the correctness of the result.

Despite these advances, adoption of formal methods for security in practice has been slow. One reason for this is that formal verification is costly. For the verification of the aforementioned seL4 kernel, for example, the proof part alone, i.e., writing proofs in sufficient detail to have them machine-checked, is reported to have required more than a person-decade of effort [KAE+10]. Such an amount of effort is only justified in few scenarios, such as building safety-critical systems. In order to make more application domains amenable to formal verification of security properties, it is necessary to make these techniques more scalable. A common approach for this is to break down the verification task into smaller problems that can be tackled more efficiently. Indeed, a refinement technique is reported to have been essential for the verification for seL4, where the security properties were proven for an abstract specification of the system and then preserved along several refinement steps to the concrete implementation. Proving the properties directly for the low-level implementation would have been intractable. While such techniques are well-established for some kinds of models and properties, there are others, in particular many notions

of secure information flow, for which refinement and (de)composition are still open challenges.

In addition to the complexity and scale of the system, the complexity of the desired security properties can become challenging, too. In many cases, it is not sufficient to enforce strict isolation between a "public" and a "private" part of the system. Instead, confidential data often needs to be shared in a controlled fashion between different users or components of the system. For example, consider a hiring workflow in a corporation with a human resources (HR) department performing interviews with applicants, and a medical department examining the physical fitness of applicants w.r.t. the job in question. In order to protect the privacy of the applicants, detailed information about their medical condition *must not* be revealed to the HR department. However, a short (binary) statement whether an applicant is fit for the job or not *must* be forwarded to the HR department as one factor contributing to the hiring decision. Hence, some information flows are desired, and actually required, while others have to be forbidden for privacy reasons. There is a large body of research on how such information flows can be analyzed and reasoned about formally. However, many existing approaches have only limited support for controlled information release, e.g. control over who may release information, but not what is released [SS09]. In the hiring workflow, for example, we would ideally like to control both, and more: the medical department (who) may release a binary health statement (what), and only once after the examination is finished (when). Combining the complexity of real-world security policies with the scalability requirements of dealing with large systems easily goes beyond what can be handled with the state of the art in verifying secure information flow.

Moreover, when putting the verification of information flow security into a larger context of system development, we find that there might be more or less subtle tensions between secure information flow and other kinds of requirements. Consider, for example, a separation of duty requirement in a scientific peer review workflow, stating that a paper must not be reviewed by anybody with a conflict of interest with one of the authors. This is not an information flow requirement, but can be modeled formally as a classical safety property: system states in which a reviewer of a paper does have a conflict of interest with it—for example, by being one of the authors—must be unreachable. However, there are also information flow requirements in this scenario, for example the reviewer anonymity requirement that authors must not learn the identities of the reviewers of their papers. If an author can somehow learn about the list of conflicts of interest with their paper—for example, by being a member of the program committee—they can use that information to narrow down the set of potential reviewers of their paper, possibly up to the point where their identities are revealed and anonymity is lifted. From an information flow perspective, this would constitute a subtle, but real potential leak that needs to be accounted for in a mathematically precise formulation of security. Doing this in a scalable way that supports expressive, real-world information flow policies is challenging.

1.2. State of the Art

There have been decades of research on formal definitions of security properties and corresponding verification techniques. We now give a brief overview of research directions from the literature. This is meant to illustrate the state of the art, and where it can be improved. We will discuss the related work in more detail in the corresponding chapters.

Many formal approaches to security are inspired by the seminal notion of noninterference. In the original setting, Goguen and Meseguer [GM82] considered automata producing a deterministic output for each input action. For a user in a given security domain, some of these actions are considered to be observable, while the others are hidden. Noninterference formalizes the idea that, whether or not any hidden actions have occurred, the output of observable actions must be unaffected. Hence, users cannot deduce information about the hidden actions that occurred from the observable outputs. In a confidentiality setting, we can use this to rule out information flows from secret inputs (hidden) to public outputs (observable). In principle, security domains can be interpreted differently, e.g. from an integrity perspective, to rule out information flows from "low" integrity inputs to "high" integrity outputs. However, all the example policies we consider in this thesis are about confidentiality requirements.

The idea behind noninterference has been transferred to various settings. For example, language-based techniques [SM03] focus on programs written in a given programming language. Typically, variables in the program source code are labeled with security levels, drawn from a lattice where greater levels represent higher confidentiality (often, the presentation of these approaches uses a two-level lattice with a "low" level L and a "high" level H, where $L \leq H$). It is assumed that an observer can inspect the part of the state containing low-confidentiality variables, and the security criterion states that running the program in observationally equivalent states is guaranteed to result in observationally equivalent states again. Hence, the values of observable variables are independent from those of unobservable variables. This means that there is no information flow from secret to public variables. Various analysis techniques have been proposed to verify this noninterference property. Volpano, Irvine, and Smith [VIS96] show the soundness of a *type system* to guarantee the absence of explicit and implicit information flows. Static analysis based on Program Dependency Graphs (PDGs) has also been used to show noninterference by checking the absence of dependencies of observable on unobservable variables. The Joanna tool [HS09], for example, uses this approach to analyze Java programs up to hundreds of thousands of lines of code. There are also dynamic enforcement techniques keeping track of security labels at runtime [RS10], implemented for example in browser plugins to enforce noninterference in JavaScript programs [BRJ+17]. These tools can be very useful to automatically verify noninterference properties of concrete programs. However, there are scenarios where language-based

techniques are not a good fit, for example when developing large, possibly heterogeneous systems where different components might eventually be implemented in different programming languages.

For such settings, approaches that operate on a higher level of abstraction are better suited. In particular, various notions of *possibilistic* information flow security, e.g. Non-Deducibility [Sut86], transfer the idea of noninterference to nondeterministic system specifications. Those are modeled in terms of the set of possible behaviors they allow. For any behavior that involves processing some specific combination of observable and secret information, there must be another possible behavior with *different* secret but the *same* observable information. Again, from the perspective of an attacker, this means that they cannot deduce from their observations which of the secrets actually occurred, since all of them are possible. Possibilistic information flow security is well-suited for dealing with system specifications, since it allows secure systems to be nondeterministic, as long as they can always make *some* nondeterministic choice that keeps confidential information protected. The MAKS framework [Man00a] unifies several classic notions of possibilistic information flow security in a common formalism, and provides conditions under which they are preserved under composition [Man02] and refinement [Man01b]. More recently, Rafnsson and Sabelfeld [RS14] present a library of composition operators for building large systems and prove compositionality results w.r.t. progress-sensitive and progress-insensitive noninterference.

However, for many realistic applications, the security properties supported by the above tools and frameworks are too strict. In particular, most of them do not support controlled information release, or *declassification*. There are different aspects of declassification that might need to be controlled in concrete applications. Sabelfeld and Sands [SS09] categorize them into the dimensions what, when, where and by whom information may be declassified. These main dimensions have further subdivisions. For example, the "what" dimension may be understood quantitatively or qualitatively. The former aims to quantify the leakage of a system, e.g. by calculating an information-theoretic measure involving the prior and posterior probability distributions of secret information from an attacker's perspective (see, for example, [Smi09] for an overview). The latter focuses less on the amount of leakage, but on what concrete details of confidential information may be released, e.g. with policies in terms of (partial) equivalence relations [SS01] specifying which potential instances of confidential information must remain indistinguishable to an observer. The situation is further complicated by the fact that real-world applications typically require declassification control in *multiple* dimensions. For example, in a scientific peer-review workflow, only the final versions of anonymized reviews ("what") may be released to authors after the program chair ("who") has finalized the decisions for each paper and initiated the notifications ("when"). Only few approaches provide the expressivity required to capture all aspects of such a policy adequately. For example, Kanav, Lammich, and Popescu [KLP14] present a security notion for I/O automata with support for controlled declassification and apply it to a conference

management system as a case study, where the above example is one of the policies that the system is verified to satisfy. However, they do not provide a compositionality result or refinement technique, making it hard to apply this security notion to large systems.

Recently, Guttman and Rowe [GR15] made a step towards further merging expressivity and scalability by presenting a flexible notion of security with declassification control, while considering systems built from communicating subsystems. They prove that, if a system does not disclose more confidential information than intended via its interface (called "cut" by Guttman and Rowe), then composing it with a system that does not generate additional confidential information preserves the guarantee of the first subsystem: An observer interacting with the composed system will still not be able to deduce more confidential information than intended. This can be seen as a compositionality result, but it remains rather specific. In particular, this result is not applicable if more than one component of the system handles confidential information.

In summary, while there are many approaches to verifying security properties in the literature, one still easily runs into limitations when trying to capture security requirements *in the large*, i.e., in realistic systems. Even if all requirements of a given application scenario can be handled by some existing approach, it is often hard to find a single approach that can capture all requirements of a given scenario adequately.

1.3. Research Questions and Scope

In this thesis we aim to improve the state of the art by combining ideas and approaches from the rich literature on secure information flow into a security notion with a novel combination of expressivity and scalability. The research questions we focus on in this thesis are:

- How can we model and verify the complex and dynamic information flow policies needed in realistic systems, such as web applications involving many users and documents?
- How can we scale up the verification of information flow security to large, realistic systems? In particular, these systems are typically developed in a modular and stepwise manner. The verification methodology needs to support this.
- How can we achieve information flow security, while at the same time satisfying security requirements other than information flow, e.g. avoiding conflicts of interest? The latter might be implemented using mechanisms that potentially leak information themselves, e.g. about the persons involved in a workflow.

> Hence, we need a way to treat information flow and other requirements in an integrated manner.

In their full generality, these questions are too broad to be answered by a single PhD thesis. It is important to narrow the scope in order to make these challenges tractable. Our first major decision to this effect is to focus our attention on *specifications* of systems on a relatively high level of abstraction. This is in line with a stepwise development approach that starts with abstract specifications and proceeds by gradual refinement. It allows us to focus our (manual) verification efforts on reasoning in terms of high-level application concepts rather than the low-level artifacts of a given programming language. Of course, the step from specification to implementation will have to be made, eventually. The case study of Chapter 4 uses the *code generator* [HN10] of the theorem prover Isabelle/HOL to generate executable Scala code directly from the verified specification. This keeps the gap between specification and implementation relatively small, although it does not close it completely: the code generator becomes part of the trusted computing base, and low-level side channels are not covered by our formal model, such as side channels involving low-level mechanisms of the runtime environment like garbage collection [PA17], or side channels involving aspects of the underlying hardware, such as caches [Aci07] or speculative execution [LSG+18, KHF+19]. Reducing this gap more formally, e.g. by complementing our verification with language-based techniques, would be very interesting, but we consider this to be beyond the scope of this thesis.

We draw many of the examples that we discuss from the domain of workflow management systems, since these often have interesting security requirements, including constraints on who may perform which activities in a workflow and which information may flow to whom. We also discuss a web application in Chapter 4 which does not involve particularly complex workflows, but subtle confidentiality requirements involving the sharing of data between the users of a social media platform.

We approach the above research questions by building on existing techniques from the literature, instead of trying to build a completely new security framework from scratch. We focus on modeling confidentiality requirements in terms of *possibilistic* information flow security, which is well-suited for nondeterministic system specifications. One aspect that this does not take into account is the *probability* distributions of secret information. Instead, a system is considered secure if an attacker cannot know protected information *with certainty*. Hence, in a system that satisfies possibilistic information flow security, an attacker might be able to deduce that one of the possible variations of the secret is much more probable than the others. Where this is a concern, the aforementioned quantitative approaches to information flow [Smi09] might be a useful alternative. However, for the scenarios we discuss in this thesis, qualitative policies on what information may be released when and to whom are more adequate. For example, in the scientific peer review workflow, such a policy specifies precisely that the complete set of final, anonymized reviews may be declassified to authors upon notification, rather than giving a quantitative bound

on how many bits of secret information may leak. Hence, we focus on possibilistic information flow control with qualitative declassification policies in this thesis.

1.4. Contributions

Our main technical contribution, presented in Chapter 3, is a compositionality result for a class of security properties that allow arbitrarily complex specifications of bounded information release. It aims to unify the strengths of existing approaches in the literature, in particular a notion of compositionality inspired by the MAKS framework [Man02], and the support for fine-grained policies of Bounded Deducibility (BD) Security [KLP14]. Thanks to the former, the resulting framework provides scalability needed to address the second research question above, and thanks to the latter, it provides expressivity needed address the first research question. We take BD Security as a starting point and extend it with compositional verification techniques. We demonstrate that BD Security can capture the security requirements of realistic systems by presenting a case study of a distributed social media platform in Chapter 4, conducted in collaboration with the co-authors of [BPPR17]. This joint work includes the development of a compositionality result that leverages a set of simplifying assumptions to make verification significantly easier. The proofs for this case study were machine-checked using the interactive theorem prover Isabelle/HOL.

Regarding the second research question, we present an application of our compositionality result to the modular development of secure workflow systems in Chapter 5. In particular, a certain notion of refinement, where an abstract execution step in a workflow specification is replaced by a refined implementation, is mapped to a composition of the overall abstract system and the concrete sub-system. We discuss this in the context of workflow management systems, but the approach could in principle be transferred to other application domains.

We use compositional reasoning to address the third research question as well. Our proposed approach is applicable to properties (other than information flow) that can be enforced using execution monitors. We analyze the information flows within such a monitor and compose the resulting information flow security property with that of the target system. As examples, we discuss the enforcement of separation of duty (as a countermeasure against conflicts of interest in workflows) as well as the enforcement of ordered delivery in an asynchronous communication platform.

1.5. Publications

Chapter 4, presenting the case study, is based on [BPPR17]. The verification of confidentiality in an earlier, non-distributed version of this system has been published

in [BPPR16] and [BPPR18]. A preliminary version of the compositional verification methodology for workflow systems (but without the notion of workflow refinement presented in Section 5.6) has been published in [BH14a]. A preliminary discussion of the compatibility of information flow security and safety properties has been published in [BH14b]. The latter two papers are based on Mantel's MAKS framework [Man00a, Man02]. The ideas and concepts presented in them have been lifted for this thesis to the BD Security framework due to Kanav, Lammich, and Popescu [KLP14].

Compared to the submitted version, this version of the thesis has been edited by adding acknowledgments and moving Section 4.3 from Chapter 3 to Chapter 4 in order to better put it into context and to make it clearer that the specialized compositionality result it describes has been developed in joint work with the co-authors of [BPPR17] as part of the case study described in Chapter 4.

1.6. Outline

The rest of this thesis is structured as follows. We begin by recalling the BD Security framework that we build upon as well as the system model and the notion of composition that we use in Chapter 2. The general compositionality result for BD Security is presented in Chapter 3, and the case study of the distributed social media platform in Chapter 4, addressing the first research question. In Chapter 5, we discuss the modular development approach for workflow systems, building upon our compositionality result, addressing the second research question. In Chapter 6, we present our approach for the integrated treatment of information flow security and other kinds of requirements, addressing the third research question. We conclude in Chapter 7 discussing possible directions for future work.

Chapter 2.

Background

This thesis builds on foundations drawn from the literature: formal models of systems, their composition, and their security properties. In this chapter, we recall the fundamental notions that we use in this thesis, beginning with the system model.

2.1. System Model

In this thesis, we do not assume that systems are specified using a particular programming language. Instead, we consider systems on a more abstract level in terms of labeled transition systems (LTSs). Transitions from one system state to another are labeled with *events*, e.g. a user providing some input. Moreover, we assume the system begins in a fixed initial state σ_0, and that there are designated sets of input and output events that form the interface of the system to other systems. Formally, a system in our sense is defined by a tuple $LTS = (St, \sigma_0, Ev, In, Out, \rightarrow)$, where

- St is the set of states with the initial state $\sigma_0 \in St$,
- Ev is the set of system events,
- $In, Out \subseteq Ev$ are disjoint subsets of input and output events, respectively, and
- $\rightarrow \subseteq St \times Ev \times St$ is the transition relation.

A *trace* is a sequence of events, written as $\langle e_1, e_2, \ldots, e_n \rangle$. We denote the empty trace as $\langle\rangle$, and the concatenation of two traces t and t' as $t \cdot t'$. When it is clear from the context that e is a single event, we write $e \cdot t$ instead of $\langle e \rangle \cdot t$ for brevity. We extend the transition relation to traces $t = \langle e_1, e_2, \ldots, e_n \rangle$ and write $\sigma_1 \stackrel{t}{\Rightarrow} \sigma_{n+1}$ if there are states σ_k such that for all $1 \leq k \leq n$, $\sigma_k \stackrel{e_k}{\rightarrow} \sigma_{k+1}$. Moreover, we write $\sigma \stackrel{t}{\Rightarrow}$ if there is some σ' such that $\sigma \stackrel{t}{\Rightarrow} \sigma'$. Conversely, we call a state σ' *reachable from* σ if there is some t such that $\sigma \stackrel{t}{\Rightarrow} \sigma'$, and we call it simply *reachable* if it is reachable from the initial state σ_0. A trace $t = \langle e_1, \ldots, e_n \rangle$ is a *valid* trace of the system LTS if it is possible from the initial state, i.e. if there is a t with $\sigma_0 \stackrel{t}{\Rightarrow}$. The *set* of valid traces is the semantic characterization of the behavior of a system that we are primarily interested in, denoted as $[\![LTS]\!] = \{t \in Ev^* \mid \sigma_0 \stackrel{t}{\Rightarrow}\}$. The length of a trace, or more generally, a sequence t is denoted as $|t|$. We denote the projection of a sequence t to elements in the set E, removing all elements that are

not in E, as $t \restriction E$. Conversely, we denote the removal of elements that *are* in E as $t \setminus E$. We write $t =_E t'$ iff $(t \restriction E) = (t' \restriction E)$. We write $t' \leq t$ if t' is a prefix of t. We use functions for mapping and filtering sequences, familiar from functional programming and defined recursively as follows:

$$\operatorname{map}(f, \langle\rangle) = \langle\rangle \qquad \operatorname{filter}(P, \langle\rangle) = \langle\rangle$$

$$\operatorname{map}(f, e \cdot t) = f(e) \cdot \operatorname{map}(f, t) \qquad \operatorname{filter}(P, e \cdot t) = \begin{cases} e \cdot \operatorname{filter}(P, t) & \text{if } P(e) \\ \operatorname{filter}(P, t) & \text{otherwise} \end{cases}$$

Moreover, the functions last, hd, and tl return the last element of a sequence, the first element of a sequence, and the remaining sequence *after* the first element, respectively. We denote the *update of a function* as $f(x := y)$; the updated function returns y for x and $f(x')$ for all $x' \neq x$. Given some relation R on a set D, the *extension of R to sequences* in D^* relates all sequences $t, t' \in D^*$ where $|t| = |t'|$ and for all $0 \leq i < |t|$, $t[i] R t'[i]$. By abuse of notation, we denote this relation using the same symbol R when it is clear from the context that we refer to sequences.

Note that we do not assume in general that the events of a system have a specific structure or meaning. Our technical definitions and results consider them as atomic, black boxes. The characteristics of events that are necessary for our discussion are captured by additional parameters of the model, e.g. the subsets of input and output events of a system, or functions mapping events to the information contained in them that is observable to an attacker. We will introduce these notions as needed while developing our theory.

Nevertheless, for illustration purposes, our examples *will* use more structured notions of events. The case study in Chapter 4 uses events comprising actions that the user has performed, together with the outputs that the system generates in response. For small examples of systems communicating with each other and with the environment, we will adopt notation used in process calculi such as CSP [Hoa85], where communication occurs by sending and receiving messages $m \in \mathbb{M}$ over channels $c \in \mathbb{C}$. Sending is represented by output events of the form $c!m$, while receiving is represented by input events of the form $c?m$. We use $\#$ as a placeholder for ! or ?, i.e. $c\#m$ represents either $c!m$ or $c?m$. We use $\bullet$ as a placeholder for messages and denote the *set* of all sending events on c as $c!\bullet = \{c!m \mid m \in \mathbb{M}\}$. The sets $c?\bullet$ and $c\#\bullet$ are defined analogously. The channel and message of an event are retrieved using functions $\operatorname{chan}(c\#m) = c$ and $\operatorname{msg}(c\#m) = m$, respectively.

As an example, consider a medical examination workflow producing a detailed medical report, and additionally a yes/no statement whether the examined person is physically fit for the specific job in question. We specify a very simple model of this workflow in a CSP-like syntax as follows (see also Figure 2.1.)

$$\begin{aligned} \mathit{Med} &= \mathit{examine}?c \to \mathit{report}?x \to \mathit{Med}'(x) \\ \mathit{Med}'(x) &= \big(\mathit{report}?x' \to \mathit{Med}'(x')\big) \\ &\quad \sqcap (\mathit{finish}?\mathrm{True} \to (\textbf{if } \mathit{decision}(x) \textbf{ then } \mathit{stmt}!\mathrm{True} \textbf{ else } \mathit{stmt}!\mathrm{False})) \end{aligned}$$

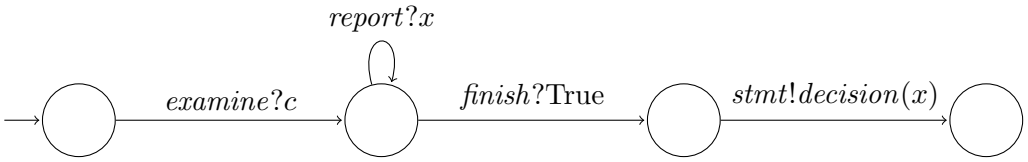

Figure 2.1.: Medical examinations process *Med*

The process *Med* begins by waiting for a request to examine some candidate c via the event *examine*?c. The physician then examines the candidate and writes a medical report with content x (modeled as *report*?x), possibly updating it multiple times. This report is kept confidential and internal to the system. Only a statement about the general physical fitness of the candidate (positive or negative) is eventually derived from the report (formalized by a function $decision\colon Report \to Bool$) after it is finished (modeled as an input on the *finish* channel) and is output to the party requesting the examination on the *stmt* channel.

Such a process induces an LTS as follows: The set of events is derived from the events occurring in the process term, and the input and output events are the subsets of receiving and sending events, respectively.

$$\begin{aligned} In_{Med} &= (examine?\bullet) \cup (report?\bullet) \cup (finish?\bullet) \\ Out_{Med} &= (stmt!\bullet) \\ Ev_{Med} &= In_{Med} \cup Out_{Med} \end{aligned}$$

The states are CSP process terms, the initial state is the term corresponding to the process *Med*, and the transition relation corresponds to a suitable labeled small-step semantics of CSP.

2.2. Security Model

Confidentiality requirements can be formalized in terms of *information flow*: which information may flow to whom, and under which conditions? Some aspects of *integrity* can be captured similarly, by controlling whether and how information from untrustworthy sources can flow into data items with high integrity requirements. We refer to [BRS10] for a discussion of different aspects of integrity, and which of them relate to information flow. The examples we discuss in this thesis are all about confidentiality. We begin by informally recalling the basic ideas underlying secure information flow, and then recall the specific notion upon which this thesis builds.

2.2.1. Information Flow Control

In order to information flow security in a formal way, we require mathematical models of what the information is we want to protect, what the powers of an attacker are, how the system behaves, and what it means for the system to be secure. For

example, in the seminal notion of noninterference [GM82], the system model is an automaton, where the attacker can perform and observe a given subset of actions. Another given set of actions is declared to be secret. The confidential information is whether any of the secret actions has actually occurred or not. A system is secure iff, for any sequence of actions *as*,

- executing *as*, and
- executing *as without* the secret actions

always yields the same outputs for observable actions. This means that the outputs of the system give an attacker no information about whether secret actions have occurred: it all looks the same to them.

In the setting of programming languages, the system is a program in a given language (with formal semantics). Information typically refers to the state of the program, for example in terms of variables and their values. Some variables are declared as confidential, and some are declared observable for the attacker. A typical formalization of security in this context requires that executing the program in two observationally equivalent states, i.e., where the observable variables have the same values, again leads to observationally equivalent states. Hence, inspecting the observable parts of the state before and after running the program gives the attacker no information about the (initial) values of secret variables. We refer to [SM03] for a survey of language-based notions of information flow security.

Possibilistic information flow security is designed to handle non-deterministic *specifications* of systems. Figure 2.2 depicts a typical model. A system is specified in terms of the set of its possible executions, called *traces*. Information is modeled by two functions O and S, extracting the observable and the secret information from a trace, respectively. A system is secure iff for any possible trace t with observation *obs* and secret *sec*, there is another possible trace with the *same* observation *obs*, but a *different* secret sec'. Hence, from observing *obs*, an attacker cannot tell for sure whether *sec* or sec' occurred; either one is possible.

There are various formulations of possibilistic information flow, differing in particular in the question how much variation is required in the secret part of traces. Nondeducibility [Sut86], for example, requires *any* secret to be possible together with *any* observation (for given, application-specific domains of secrets and observations, respectively). The MAKS framework [Man00a] is more specific. Observations and secrets are the sequences of visible or observable events in a trace, respectively. The required variations of the secret are specified in terms of Basic Security Predicates (BSPs). For example, the BSP D requires that the deletion of confidential in a trace is possible without changing the observation. Similar to noninterference, this keeps the occurrence of confidential events secret from an observer. Conversely, the BSP I requires the insertion of confidential events, keeping the *non*-occurrence of confidential events secret. There are variations of these BSPs, and several classical notions of possibilistic information flow security can be expressed in terms of combinations of BSPs.

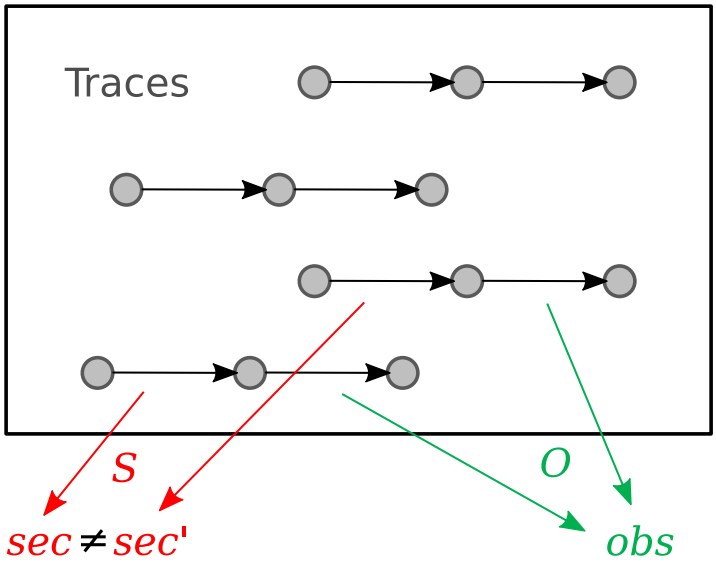

Figure 2.2.: Possibilistic information flow security

In this thesis, we build upon *Bounded Deducibility (BD) Security* [KLP14], which is a simple but powerful generalization of Nondeducibility. It allows formulating very fine-grained and application-specific requirements on the variation of secrets, in terms of "declassification bounds." While Nondeducibility allows *no* information flow from secrets to observers, BD Security enforces *bounds* on what an observer may learn about the secret. We now recall the formal definition of this notion.

2.2.2. Bounded Deducibility Security

Bounded Deducibility Security is a generalization of Sutherland's Nondeducibility [Sut86]. A system is viewed as a set of possible execution histories (called "possible worlds" by Sutherland). The observation function O maps such a trace to the information it provides to an observer of the system, while the secrets function S maps a trace to the secret information contained in it. Nondeducibility requires that the system must be able to produce *every combination* of observation and secrets. Hence, observers cannot deduce anything about the secrets from their observations. Even though this is a simple and elegant property, it is too restrictive in practice; it is usually necessary to allow *some* flows of information. For example, in a hiring workflow, the result of a medical examination may have to be released to the human resources department, but any further medical details have to be kept confidential.

Bounded Deducibility Security [KLP14] therefore adds another parameter, called the *declassification bound* B. It is a binary relation on secrets that relates those secrets that must be indistinguishable for the observer. It specifies a lower bound on the uncertainty of the observer about the secret, or equivalently, an upper bound on the declassification allowed for the system.

Definition 2.1. Let *Sys* be some kind of system (possibly, but not necessarily an LTS) characterized by its set of possible behaviors $[\![Sys]\!] \subseteq \mathsf{Beh}$, let $O\colon \mathsf{Beh} \to \mathsf{O}$ and $S\colon \mathsf{Beh} \to \mathsf{S}$ be observation and secret extracting functions, respectively, and let $B \subseteq \mathsf{S} \times \mathsf{S}$ be a binary relation on secrets.

The system *satisfies (abstract) BD Security w.r.t. O, S, and B* iff, for all $t \in [\![Sys]\!]$ and $s' \in \mathsf{S}$,

$$(S(t), s') \in B \longrightarrow (\exists t' \in [\![Sys]\!].\, O(t') = O(t) \wedge S(t') = s')$$

Intuitively, BD security requires that, if $(s, s') \in B$, then observers cannot distinguish the secret s from s' via their observations—if s is possible together with a given observation, then so must be s'. Classical nondeducibility corresponds to B being the total relation $\mathsf{S} \times \mathsf{S}$—the observer can then deduce *nothing* about the secret. Smaller relations B mean that an observer may deduce some information about the secret, but nothing beyond B—for example, if B is an equivalence relation, then the observer may deduce the equivalence class of the secret, but not the concrete value within the equivalence class.

2.2.3. Instantiation for Labeled Transition Systems

The above, abstract definition of BD Security leaves open what system behaviors, observations, and secrets are. In this thesis, we consider labeled transition systems and their traces of events, and we focus on *sequences* of observations and secrets produced by the *individual events* in a trace. Hence, the abstract observation and secret domains O and S of Definition 2.1 are instantiated with Obs^* and Sec^*, respectively, where Obs and Sec represent observable and secret information contained in individual events. This comes quite naturally in the scenarios we have studied: For example, if the observers are a set of users of the system, then the observable part of a trace is the sequence of events representing the interaction of those users with the system. Similarly, if the system is supposed to keep a set of documents confidential, then the sequence of updates to those documents constitute the secret part of a trace. We capture the observational and secrecy setup in a given application scenario formally in terms of a *view* onto the system.

Definition 2.2. A *view on Ev* is a tuple $\mathcal{V} = (Ev^{obs}, getObs, Ev^{sec}, getSec)$ with

$$\begin{array}{ll} Ev^{obs} \subseteq Ev & getObs : Ev^{obs} \to \mathsf{Obs} \\ Ev^{sec} \subseteq Ev & getSec : Ev^{sec} \to \mathsf{Sec} \end{array}$$

The observable and secret parts of *traces* $t \in Ev^*$ w.r.t. $\mathcal{V}$ are defined as

$$\begin{aligned} O_{\mathcal{V}}(t) &= \mathrm{map}(getObs, t \upharpoonright Ev^{obs}) \\ S_{\mathcal{V}}(t) &= \mathrm{map}(getSec, t \upharpoonright Ev^{sec}) \end{aligned}$$

LTS is *BD secure w.r.t. $\mathcal{V}$ and B* iff for all $t \in [\![LTS]\!]$ and $s' \in \mathsf{Sec}^*$,

$$(S_{\mathcal{V}}(t), s') \in B \longrightarrow (\exists t' \in [\![LTS]\!].\, O_{\mathcal{V}}(t') = O_{\mathcal{V}}(t) \wedge S_{\mathcal{V}}(t') = s')$$

The observation extracting function *getObs* specifies *which aspects* of events are observable. The value of $getObs(e)$ models the information that observers get when the event e occurs. If $getObs(e) = getObs(e')$, then the events e and e' are indistinguishable for an observer, but may differ in aspects that are not observable. Similarly, the secret extracting function *getSec* allows us to specify which aspects of events are secret. Again, this function does not need to be injective. Events that contain the same observable and/or secret information can be grouped into equivalence classes as follows; these equivalence relations will be useful in Chapter 3 for formalizing constraints for compositionality.

Definition 2.3. Two events e, e' are

- *observation-equivalent for* $\mathcal{V}$, denoted $e \approx^{obs}_{\mathcal{V}} e'$, if $e \in Ev^{obs}_{\mathcal{V}}$, $e' \in Ev^{obs}_{\mathcal{V}}$, and $getObs(e) = getObs(e')$,
- *secret-equivalent for* $\mathcal{V}$, denoted $e \approx^{sec}_{\mathcal{V}} e'$, if $e \in Ev^{sec}_{\mathcal{V}}$, $e' \in Ev^{sec}_{\mathcal{V}}$, and $getSec(e) = getSec(e')$, and
- $\mathcal{V}$*-equivalent*, denoted $e \approx_{\mathcal{V}} e'$, if
  - $e \in Ev^{obs} \cup Ev^{sec}$,
  - $e \in Ev^{obs}_{\mathcal{V}} \longleftrightarrow e' \in Ev^{obs}_{\mathcal{V}}$, and $e \in Ev^{obs}_{\mathcal{V}}$ implies $e \approx^{obs}_{\mathcal{V}} e'$, and
  - $e \in Ev^{sec}_{\mathcal{V}} \longleftrightarrow e' \in Ev^{sec}_{\mathcal{V}}$, and $e \in Ev^{sec}_{\mathcal{V}}$ implies $e \approx^{sec}_{\mathcal{V}} e'$.

Note that we don't require Ev^{obs} and Ev^{sec} to be disjoint; there might be events that are partially observable *and* contain secret information, as in the following example.

2.2.4. Example: Progress-Sensitive Noninterference

Consider a system that communicates on channels that can be partitioned as follows:

- a set L of "low" channels that are fully observable,
- a set M of "medium" channels where the occurrence of messages is observable, but the content is protected via encryption, and
- a set H of "high" channels that are fully invisible.

Assume we want to keep the *content* of input messages on channels in M confidential, and both the *occurrence and content* of input messages on channels in H. We choose to treat *output* messages as non-confidential in this example. In many cases, this is adequate; in particular, if the outputs depend deterministically on the inputs, then only the latter are the source of secret information, and treating outputs as confidential would be redundant. In other cases, outputs *do* carry confidential information, and we will consider such cases in later chapters. However, this requires us to specify any dependencies of outputs on inputs in the declassification bound,

in order to obtain a security property that can be satisfied realistically. For this simple example, we focus on the confidentiality of *inputs* only and allow the system to choose its outputs on confidential channels freely.

We define the view

$$\mathcal{V}_{L,M,H} = (Ev^{obs}_{L,M}, getObs_{L,M}, Ev^{sec}_{H,M}, getSec_{H,M})$$

as follows. Events on channels in L are observable, input events on channels in H are secret, output events on channels in H are treated as neutral (neither observable nor secret), and events on channels in M are *both* (partially) observable and secret.

$$Ev^{obs}_{L,M} = \left(\bigcup_{c \in L \cup M} c\#\bullet\right) \qquad Ev^{sec}_{H,M} = \left(\bigcup_{c \in H} c?\bullet\right) \cup \left(\bigcup_{c \in M} c\#\bullet\right)$$

Assuming that observers cannot break cryptography, they cannot see the contents of messages on encrypted channels $c \in M$, but we assume that they are able to observe the *occurrence* of message transmissions. We model this by defining *getObs* so that it identifies all events of the form $c\#\bullet$ by returning the same value $c\#\mathtt{d}$ for some dummy message $\mathtt{d}$, while it is the identity for events on other observable channels $c' \in L$ that are not encrypted. For the secret producing function *getSec*, we use the identity on input events, specifying that the full content of messages on encrypted or invisible channels is secret information. Output events are not treated as confidential in our example, as discussed above. Hence, we ignore their content by replacing it with a dummy value in *getSec*, analogous to *getObs*.

$$getObs_{L,M}(e) = \begin{cases} c'\#m & \text{if } e = c'\#m \wedge c' \in L \\ c\#\mathtt{d} & \text{if } e = c\#m \wedge c \in M \end{cases}$$

$$getSec_{H,M}(e) = \begin{cases} c?m & \text{if } e = c?m \\ c!\mathtt{d} & \text{if } e = c!m \end{cases}$$

We only allow the system to declassify information that is already observable: when and on which channels *encrypted* inputs are received. Formally, we define the declassification bound $B_{L,M,H} \subseteq (Ev^{sec}_{H,M})^* \times (Ev^{sec}_{H,M})^*$ with

$$(sl, sl') \in B_{L,M,H} \longleftrightarrow O_{\mathcal{V}_{L,M,H}}(sl) = O_{\mathcal{V}_{L,M,H}}(sl')$$

Note that $O_{\mathcal{V}_{L,M,H}}$, applied to a sequence of secret events, selects only events on channels in M (the intersection of observable and secret events) and maps them via $getObs_{L,M}$, replacing their content with a dummy value. Hence, the bound $B_{L,M,H}$ relates arbitrary sequences of secret inputs, provided that the communication patterns (without content) on encrypted channels look the same to the observers.

The resulting property is similar to Rafnsson and Sabelfeld's Progress-Sensitive Noninterference (PSNI) [RS14].[1] Intuitively, PSNI demands that secret inputs on

[1] It seems that progress-*insensitive* noninterference (PINI) as defined in [RS14] cannot be directly

high channels can be inserted and deleted arbitrarily, and inputs on encrypted channels can be replaced arbitrarily, both without interfering with the observations. Hence, while the occurrence and relative timing of certain confidential communication events is declassified, the observer learns nothing about the contents of confidential inputs.

Note, however, that the above BD Security property is only a simplified approximation of PSNI for illustration purposes. In particular, it is not compositional, while the PSNI property presented in [RS14] *can* be composed. In Section 3.4.2, we will strengthen the above property in order to make it compositional.

2.2.5. Example: Medical Examination Workflow

As an example of a system with security requirements that go beyond noninterference, consider again the medical examination workflow of Section 2.1. We assume that the details of the resulting medical report are to be kept confidential, while a yes/no statement whether the examined person is physically fit for the given job is to be released to an observing party at the end. In this setting, the system does not satisfy classical noninterference. It is not possible to delete all confidential inputs, because the system insists on at least one input of a medical report before a statement is output. It is not always possible to insert confidential inputs, either (as, for example, Rafnsson and Sabelfeld's security properties [RS14] demand for channels with confidential content). In particular, inserting an update to a medical report at the end of the examination is not possible if its content is not compatible with the observable final statement. Hence, there is an information flow: If the statement is negative, then the observer can deduce that the final version of the medical report *cannot* have read "No health issues found."

However, this information flow is actually desired. We just have to make sure that the final decision is the *only* information that flows about the content of the report. We formalize this as a BD security property as follows. Let $\approx_{Med}$ be an equivalence relation on medical reports such that $x \approx_{Med} x'$ iff $\mathit{decision}(x) = \mathit{decision}(x')$, and let $\mathcal{V}_{Med}$ be a view with

$$\begin{aligned} Ev^{obs}_{Med} &= (\mathit{examine}?\bullet) \cup (\mathit{stmt}!\bullet) & \mathit{getObs}_{Med} &= \mathrm{id} \\ Ev^{sec}_{Med} &= (\mathit{report}?\bullet) & \mathit{getSec}_{Med} &= \mathrm{id} \end{aligned}$$

The BD security property with $\mathcal{V}_{Med}$ and $B_{Med} \subseteq (Ev^{sec}_{Med})^* \times (Ev^{sec}_{Med})^*$ such that

$$\begin{aligned} (sl, sl') \in B_{Med} \longleftrightarrow (sl = \langle\rangle \longleftrightarrow sl' = \langle\rangle) \wedge \\ (sl \neq \langle\rangle \longrightarrow \mathrm{msg}(\mathrm{last}(sl)) \approx_{Med} \mathrm{msg}(\mathrm{last}(sl))) \end{aligned}$$

expressed as an instance of Definition 2.2. PINI allows either the original trace t or its replacement t' to diverge, so it only requires the value and observation of t to be prefixes of those of t' (or vice versa). This suggests that BD Security could be generalized by making it parametric in the relations that are used to compare secrets and observations, instead of demanding equality. We leave this as future work.

formalizes the requirement that observers can only deduce two pieces of information about the sequence of medical reports:

- The non-occurrence of inputs: Before an *examine* event has occurred, an observer knows that the process has not accepted any reports yet.
- The equivalence class of the last version of the report that was input: This is declassified in the final *stmt* event.

The system *Med* is indeed BD secure w.r.t. $\mathcal{V}_{Med}$ and B_{Med}: Given a trace with a non-empty sequence of report inputs, we can replace the inputs with any other sequence, as long as the final inputs are equivalent. The observation, in particular the final statement sent on the *stmt* channel, will be the same. How can we verify that a system satisfies such a BD security property? Kanav, Lammich and Popescu present an unwinding technique for BD security in [KLP14]. In Appendix B, we sketch the application of the technique to this example, as well as an extension for unwinding the variants of BD Security we will define in Chapter 3.

2.3. Composition of Labeled Transition Systems

As a motivating example for the use of composition operators, consider the composition of *Med* with a system *Rev*, depicted in Figure 2.3, representing an application reviewing process. Its behavior is defined as follows.

$$\begin{aligned} \mathit{Rev} &= \mathit{review}?c \rightarrow (\mathit{Examine}(c) \sqcap (\tau \rightarrow \mathit{reject}!c)) \\ \mathit{Examine}(c) &= \mathit{examine}!c \rightarrow \mathit{stmt}?s \\ &\quad \rightarrow (\textbf{if } s \textbf{ then } \mathit{shortlist}!c \textbf{ else } \mathit{reject}!c) \end{aligned}$$

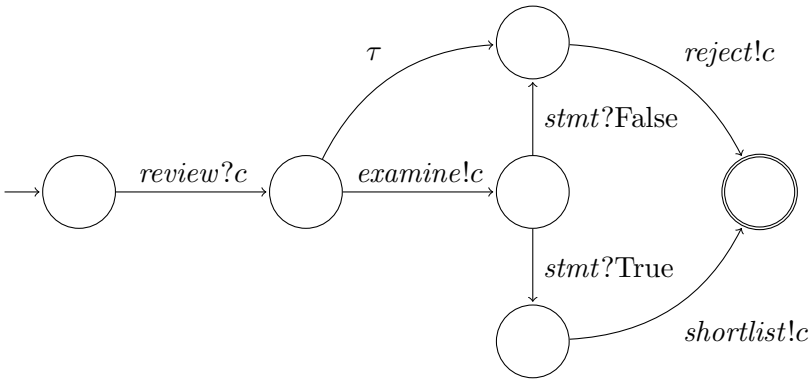

Figure 2.3.: Application reviewing process *Rev*

After a request to review the application of a candidate c comes in, the application is considered and either rejected or, in case of a positive review, the candidate is sent to a medical examination. Depending on the result of the examination, the

review process for this candidate is either finished negatively or the candidate is shortlisted. This process can be modeled by composing the *Rev* and *Med* systems and synchronizing them on communication events.

Now suppose that the position to be filled has very specific physical requirements, so the employer demands two independent medical examinations. We will model this workflow by composing two copies of the medical examination process and synchronizing them with the reviewing process. First, however, we introduce the semantics of the composition operator we use in this thesis.

2.3.1. Binary composition

There are various notions of composition in the literature with subtle and less subtle differences, e.g. synchronous vs. asynchronous communication. In CSP [Hoa85], processes interact via matching communication events: one process can only emit a sending event $c!v$ on a shared channel c if the receiving process is ready to accept $c?v$ and vice versa. In the trace of the composed process, the event $c!v$ is retained to record successful synchronization. In CCS [Mil80], on the other hand, synchronization is hidden from the environment as a silent τ transition. Moreover, processes don't *have to* synchronize: each of them may always make individual process and communicate with the environment instead of with the other process. In ACP [BK85], synchronization is parameterized with a composition function $(a|b)$ that specifies *whether* two events a and b synchronize, and *what* the result of the synchronization is, e.g. a different event c.

We follow an approach similar to ACP and assume a synchronization relation

$$\| \subseteq (Out_1 \cup Out_2) \times (In_1 \cup In_2)$$

specifying, for two systems LTS_1 and LTS_2, which output events of one system can synchronize with which input events of the other system. This allows us to use not only events of the form $c!v$ and $c?v$ for communication, but also more customized events. Moreover, it allows us to specify a *many-to-many* relationship between synchronizing events. For example, in the case study in Chapter 4, the events causing communication also contain local information, e.g. the password of the user initiating the communication, which is not sent over the network and is ignored for synchronization. Hence, multiple sending events (with different local passwords) can synchronize with the same receiving event.[2]

Given a synchronization relation, the two systems can either synchronously[3] perform communication events related by $\|$, or they can make individual progress by

[2] An alternative solution is to split the event into two: the local initiation of communication and the sending of the message. However, in the case study, we found it more convenient to treat it as one event. The synchronization relation gives us the freedom to choose either of the modeling approaches.

[3] *Asynchronous* communication can be modeled in this paradigm by adding buffer components that store messages and deliver them at a later point in time.

performing *local* events that do not require interaction with the other system. We denote the set of interface events of LTS_i that *do* require interaction as

$$If_i = \{e \in Ev_i \mid \exists e' \in Ev_j.\, e \parallel e' \vee e' \parallel e\}$$

A local event of LTS_i is an event $e \in Ev_i \setminus If_i$.

We construct an LTS representing the composed system with a state space that is the product of the state spaces of the components. Its event set is the disjoint union of the individual event sets and the set of pairs of events:

- $(1, e_1)$ represents a local event e_1 performed by LTS_1,
- $(2, e_2)$ represents a local event e_2 performed by LTS_2, and
- (e_1, e_2) represents a communication of LTS_1 performing e_1 and LTS_2 performing e_2.

Note that, if a more customized structure of events is desired for the composed system, this can be achieved by a translation of events after composition. In Section 4.3.2, we discuss the security-preserving translation of events, states, observations, and secrets.

Definition 2.4. Let $\parallel \subseteq (Out_1 \cup Out_2) \times (In_1 \cup In_2)$ be a synchronization relation between LTS_1 and LTS_2. The *(binary) composition of LTS_1 and LTS_2 w.r.t.* $\parallel$ is an $LTS = (St, \sigma_0, Ev, In, Out, \rightarrow)$ with

$$\begin{aligned}
St &= St_1 \times St_2 \text{ and } \sigma_0 = (\sigma_{0,1}, \sigma_{0,2}) \\
Ev &= \{(1, e_1) \mid e_1 \in Ev_1 \setminus If_1\} \cup \\
&\quad \{(2, e_2) \mid e_2 \in Ev_2 \setminus If_2\} \cup \\
&\quad \{(e_1, e_2) \mid e_1 \in If_1 \wedge e_2 \in If_2 \wedge (e_1 \parallel e_2 \vee e_2 \parallel e_1)\} \\
In &= \{(1, e_1) \mid e_1 \in In_1 \setminus If_1\} \cup \\
&\quad \{(2, e_2) \mid e_2 \in In_2 \setminus If_2\} \\
Out &= \{(1, e_1) \mid e_1 \in Out_1 \setminus If_1\} \cup \\
&\quad \{(2, e_2) \mid e_2 \in Out_2 \setminus If_2\}
\end{aligned}$$

and the transition relation $\rightarrow$ defined inductively in Figure 2.4.

The *set of valid traces* of LTS can also be characterized as the union of compositions of valid traces of LTS_1 and LTS_2. For this purpose, we lift the synchronization of individual events to the synchronization of traces and define an operator $\parallel : Ev_1^* \times Ev_2^* \rightarrow 2^{Ev^*}$ inductively via the rules in Figure 2.5. For simplicity, we use the same symbol $\parallel$ for trace composition; it will be clear from the context whether we compose events or traces. The set of traces of the composed system LTS can now be written as

$$[\![LTS]\!] = \{t \mid \exists t_1 \in [\![LTS_1]\!], t_2 \in [\![LTS_2]\!].\, t \in t_1 \parallel t_2\}$$

$$\textsc{Local}_1\ \frac{\sigma_1 \xrightarrow{e_1}_1 \sigma_1' \qquad e_1 \notin \mathit{If}_1}{(\sigma_1, \sigma_2) \xrightarrow{(1,e_1)} (\sigma_1', \sigma_2)} \qquad \textsc{Local}_2\ \frac{\sigma_2 \xrightarrow{e_2}_2 \sigma_2' \qquad e_2 \notin \mathit{If}_2}{(\sigma_1, \sigma_2) \xrightarrow{(2,e_2)} (\sigma_1, \sigma_2')}$$

$$\textsc{Comp}\ \frac{\sigma_1 \xrightarrow{e_1}_1 \sigma_1' \qquad \sigma_2 \xrightarrow{e_2}_2 \sigma_2' \qquad e_1 \parallel e_2 \vee e_2 \parallel e_1}{(\sigma_1, \sigma_2) \xrightarrow{(e_1,e_2)} (\sigma_1', \sigma_2')}$$

Figure 2.4.: Transition relation for binary composition

$$\textsc{Nil}\ \frac{}{\langle\rangle \in \langle\rangle \parallel \langle\rangle} \qquad \textsc{Comp}\ \frac{t \in t_1 \parallel t_2 \qquad e_1 \parallel e_2 \vee e_2 \parallel e_1}{(e_1, e_2) \cdot t \in (e_1 \cdot t_1) \parallel (e_2 \cdot t_2)}$$

$$\textsc{Local}_1\ \frac{t \in t_1 \parallel t_2 \qquad e_1 \notin \mathit{If}_1}{(1, e_1) \cdot t \in (e_1 \cdot t_1) \parallel t_2} \qquad \textsc{Local}_2\ \frac{t \in t_1 \parallel t_2 \qquad e_2 \notin \mathit{If}_2}{(2, e_2) \cdot t \in t_1 \parallel (e_2 \cdot t_2)}$$

Figure 2.5.: Trace composition

Moreover, note that traces of *LTS* can be uniquely decomposed into its component traces. We define the projection of a composed trace t to the first or second component, denoted $t \restriction 1$ or $t \restriction 2$, respectively, recursively as $\langle\rangle \restriction i = \langle\rangle$ and

$$(e \cdot t) \restriction i = \begin{cases} e_i \cdot (t \restriction i) & \text{if } e = (i, e_i) \\ e_1 \cdot (t \restriction i) & \text{if } e = (e_1, e_2) \wedge i = 1 \\ e_2 \cdot (t \restriction i) & \text{if } e = (e_1, e_2) \wedge i = 2 \\ (t \restriction i) & \text{if } e = (j, e_j) \text{ with } j \neq i \end{cases}$$

This allows the following characterization of $[\![\mathit{LTS}]\!]$.

Lemma 2.1. *Let LTS be the composition of LTS_1 and LTS_2 w.r.t. $\parallel$. For all $t \in Ev^*$,*

$$t \in [\![\mathit{LTS}]\!] \longleftrightarrow (t \restriction 1) \in [\![\mathit{LTS}_1]\!] \wedge (t \restriction 2) \in [\![\mathit{LTS}_2]\!]$$

For proofs of the lemmas and theorems in this thesis, see Appendix C.

In summary, we use a composition operator that is parametric in the synchronization relation $\parallel$ and does not make any assumptions about the events themselves. The advantage of this approach is that it is very flexible, as we will see in later chapters. One disadvantage is that, due to the structure of composed events, this notion of composition is neither commutative nor associative in general. However, we will present a translation theorem in Section 4.3.2 that allows us to change the structure of events (e.g., after composition) while preserving security, if certain conditions are

met. We will use this in Sections 4.3 and 5.5 to prove n-ary compositionality results for BD Security w.r.t. the following notion of n-ary composition.

2.3.2. n-ary Composition

For modeling a network of components, we use a *family* of synchronization relations $\|_{i,j}$ specifying the possible synchronizations between output events of component i and input events of component j. Analogously to the binary case, we denote the set of interface events of component i with component j as

$$If_{i,j} = \{e \in Ev_i \mid \exists e' \in Ev_j.\, e \|_{i,j} e' \vee e' \|_{j,i} e\}$$

and the set of *all* interface events of i as $If_i = \bigcup_{j \in N \setminus \{i\}} If_{i,j}$. The state of the network is a function mapping a component identifier to its local state. The events are similar to the binary case, but we now always include the identifiers of the participating components, e.g. (i, e_i, j, e_j) represents a synchronization between component i performing e_i and component j performing e_j.

Definition 2.5. Let Net be a family $(LTS_i)_{i \in N}$ of LTSs indexed by a finite set N with

$$LTS_i = (St_i, \sigma_{0,i}, Ev_i, In_i, Out_i, \rightarrow_i)$$

Let $(\|_{i,j})_{i,j \in N}$ be a family of synchronization relations with $\|_{i,j} \subseteq Out_i \times In_j$, and let $St_N = \bigcup_{i \in N} St_i$ be the union of state spaces of the systems in Net. We model the global state of the network as a function $\sigma \colon N \to St_N$ with $\sigma(i) \in St(i)$. Hence, the global state space St is the *set* of all those functions, $N \to St_N$.

The *composition of Net w.r.t.* $(\|_{i,j})_{i,j \in N}$ is an $LTS = (St, \sigma_0, Ev, In, Out, \rightarrow)$ with

$$\begin{aligned}
St &= (N \to St_N) \text{ and } \sigma_0(i) = \sigma_{0,i} \\
Ev &= \bigcup_{i \in N} \{(i, e_i) \mid e_i \in Ev_i \setminus If_i\} \cup \\
&\quad \bigcup_{i,j \in N} \{(i, e_i, j, e_j) \mid e_i \in If_i \wedge e_j \in If_j \wedge e_i \|_{i,j} e_j\} \\
In &= \bigcup_{i \in N} \{(i, e_i) \mid e_i \in In_i \setminus If_i\} \\
Out &= \bigcup_{i \in N} \{(i, e_i) \mid e_i \in Out_i \setminus If_i\}
\end{aligned}$$

and the transition relation $\rightarrow$ defined inductively in Figure 2.6.

Again, there is an alternative perspective using a trace composition operator. Let $Ev_N = \bigcup_{i \in N} Ev_i$ be the union of the event sets of all nodes in the network. We model a collection of traces of the individual nodes in the network as a function from N to Ev_N^*. The trace composition operator

$$\|_N : (N \to Ev_N^*) \to 2^{Ev^*}$$

takes such a collection of component traces of the components and returns the set of possible composed traces according to the rules in Figure 2.7.

In this context, the projection of a composed trace to a given component i is defined, similarly to the binary case, as $\langle\rangle \restriction i = \langle\rangle$ and

$$(e \cdot t) \restriction i = \begin{cases} e_i \cdot (t \restriction i) & \text{if } e = (i, e_i) \\ e_i \cdot (t \restriction i) & \text{if } e = (i, e_i, j, e_j) \vee e = (j, e_j, i, e_i) \\ t \restriction i & \text{otherwise} \end{cases}$$

This leads to the following characterization of $[\![LTS]\!]$ in terms of the valid traces of the components.

Lemma 2.2. *Let LTS be the composition of composition of Net w.r.t.* $(\ \|_{i,j}\ )_{i,j \in N}$. *For all* $t \in Ev^*$,

$$t \in [\![LTS]\!] \longleftrightarrow (\forall i \in N.\ (t \restriction i) \in [\![LTS_i]\!])$$

For example, we can now construct the workflow system described above via composition as follows. Let Med_1 and Med_2 be two copies of the process *Med* where the names of their channels are tagged with 1 and 2, respectively.[4] The parallel composition of these processes models the two independent medical examinations. As the "glue" to the reviewing process, we define the following processes:

$$\begin{aligned} \mathit{Split} &= \mathit{examine}?c \to \mathit{examine}_1!c \to \mathit{examine}_2!c \\ \mathit{Join} &= \mathit{stmt}_1?d_1 \to \mathit{stmt}_2?d_2 \\ &\qquad \to (\textbf{if } d_1 \wedge d_2 \textbf{ then } \mathit{stmt}!\mathrm{True} \textbf{ else } \mathit{stmt}!\mathrm{False}) \end{aligned}$$

The overall example workflow is the composition of the processes $\{Rev, Split, Med_1, Med_2, Join\}$ where the synchronization relations simply relate sending events of one component and the corresponding receiving events of the other component:

$$e_i \ \|_{i,j}\ e_j \longleftrightarrow (\exists c, m.\ e_i = c!m \wedge e_j = c?m \wedge e_i \in Out_i \wedge e_j \in In_j)$$

The wiring of the channels between the components is depicted in Figure 2.8.

In order for such a composition to be well-defined, we require that synchronization is *pairwise dedicated*: for each communication event, there must be a unique component in the network which is the target of the communication.

Definition 2.6. We say the network *Net* is *composable w.r.t.* $(\ \|_{i,j}\ )_{i,j \in N}$ iff

- $e_i \ \|_{i,j}\ e_j$ and $e_i \ \|_{i,k}\ e_k$ imply $k = j$, and
- $e_i \ \|_{i,j}\ e_j$ and $e_m \ \|_{m,j}\ e_j$ imply $m = i$.

[4] Formally, we define the renamings of channels using bijections on events $r_i(c\#m) = c_i\#m$ and lift them to systems $r_i(LTS) = (St, \sigma_0, r_i(Ev), r_i(In), r_i(Out), \to_r)$ with $\sigma \xrightarrow{e}_r \sigma'$ iff $\sigma \xrightarrow{r^{-1}(e)} \sigma'$.

$$\textsc{Local}\ \frac{\sigma(i) \xrightarrow{e_i}_i \sigma_i' \qquad e_i \notin \mathit{If}_i}{\sigma \xrightarrow{(i,e_i)} \sigma(i := \sigma_i')} \qquad \textsc{Comp}\ \frac{\sigma(i) \xrightarrow{e_i}_i \sigma_i' \qquad \sigma(j) \xrightarrow{e_j}_j \sigma_j' \qquad e_i \parallel_{i,j} e_j}{\sigma \xrightarrow{(i,e_i,j,e_j)} \sigma(i := \sigma_i', j := \sigma_j')}$$

Figure 2.6.: Transition relation for n-ary composition (where i, j with $i \neq j$ range over network nodes)

$$\textsc{Nil}\ \frac{}{\langle\rangle \in \parallel_N (i \in N \mapsto \langle\rangle)} \qquad \textsc{Local}\ \frac{t \in \parallel_N ts \qquad e_i \notin \mathit{If}_i}{(i, e_i) \cdot t \in \parallel_N ts(i := e_i \cdot ts(i))}$$

$$\textsc{Comp}\ \frac{t \in \parallel_N ts \qquad e_i \parallel_{i,j} e_j}{(i, e_i, j, e_j) \cdot t \in \parallel_N ts(i := e_i \cdot ts(i), j := e_j \cdot ts(j))}$$

Figure 2.7.: n-ary trace composition

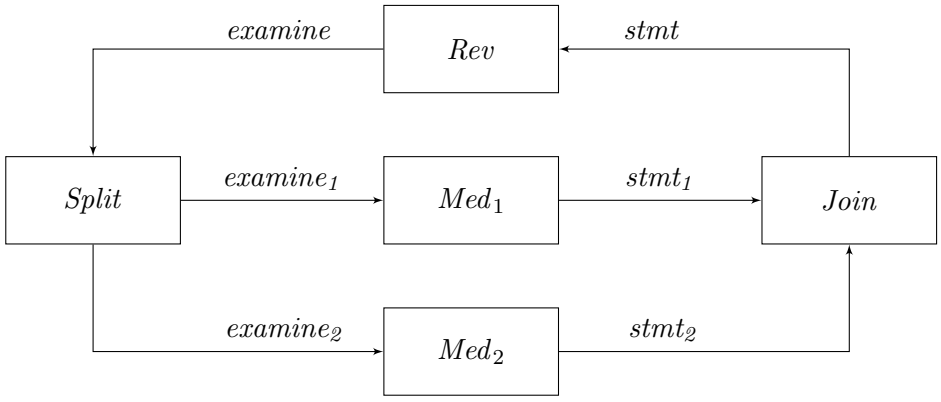

Figure 2.8.: Wiring of components in the hiring example

For the example composition, this holds: As can be seen in Figure 2.8, each channel is used for communication between exactly two components. The only exceptions (not depicted in Figure 2.8) are the channels used for communication with the *environment*, not between the components: the channels $report_1$ and $report_2$ are used by Med_1 and Med_2, respectively, to receive updates to the medical reports as input, and the process *Rev* receives the initial request for a candidate review on the channel *review* and outputs the result to the environment on the channels *reject* and *shortlist*.

Chapter 3.

A Compositionality Result for Bounded Deducibility Security

It is well-known that information flow security is not compositional in general. This was recognized already for early security notions [McC87]. Note that possibilistic information flow security typically considers *open* systems, whose environments can behave in *any* possible way. When composing such a system with another one, each of them becomes (part of) the other's environment. This corresponds to a *refinement* of the previously unconstrained environments. It is another well-known fact that possibilistic information flow security is not preserved under refinement in general. The reason for this is that the former, in particular BD Security, demands the possibility of certain traces, while refinement reduces the set of possible traces. There is a clear tension between those two requirements.

One approach to solve this problem is to strengthen the security properties of the components so that they are preserved under arbitrary refinements of the environment. For example, Non-Deducibility under Composition [FG93] only considers a system to be secure if it keeps secret information confidential in any environment it is composed with. Another approach is to consider not arbitrary environments, but only the ones the system might actually be composed with. The MAKS framework, for example, focuses on the interface between given systems, and identifies additional, interface-related security requirements such that, if the systems satisfy those, it is guaranteed that their security properties can be composed. The advantage is that this gives some flexibility for application-specific trade-offs in distributing proof obligations. Weakening the security guarantees of one component can, to some degree, be compensated by strengthening the guarantees of the other component(s).

In this chapter, we follow the latter approach. For BD Security, the problem is exacerbated by the high expressivity that the framework provides for specifying observations, secrets, and declassification bounds. In the following sections, we will discuss both examples and counterexamples for the compositionality of BD Security. The challenges that we face with these examples will guide us towards generic compositionality results for BD Security. We formulate an abstract version of such a result for arbitrary composition operators, using a notion of well-behavedness as the main condition for compositionality. We instantiate this to the composition of labeled transition systems as introduced in Chapter 2. We focus on binary composition first, but will consider n-ary composition in Chapters 4 and 5.

3.1. Towards Composing BD Security Properties

The first step on our path to composing BD Security properties of labeled transition systems is to construct a security view for the composed system. We define a canonical composition of the local views of the components in a straightforward way, similar to the composition of systems.

- Local events are observable iff they are observable in the corresponding local view; the local observations are annotated with 1 or 2, identifying the component that produced them.
- Synchronization events are observable iff both of the participating events are observable in the corresponding local view. We will rule out the case that only one of them is observable by making an explicit assumption that synchronizing events are either both observable or both non-observable. We allow the merging of the two local observations via an operator $*_O \colon \mathsf{Obs}_1 \times \mathsf{Obs}_2 \to \mathsf{Obs}$.

Secrets are lifted analogously, with a secret merging operator $*_S \colon \mathsf{Sec}_1 \times \mathsf{Sec}_2 \to \mathsf{Sec}$. Note that we allow pairs of synchronizing events where only one of them is secret; the composed event is then considered to be non-confidential. If we want a synchronization event to be secret for the composed system, we declare *both* participating events to be secret for the components.

Definition 3.1. The *composition of* $\mathcal{V}_1$ *and* $\mathcal{V}_2$ *w.r.t.* $\parallel$, $*_O$, *and* $*_S$ is the view

$$\mathcal{V} = (Ev^{obs}, getObs, Ev^{sec}, getSec)$$

with

$$\begin{aligned}
Ev^{obs} = {} & \{(1, e_1) \mid e_1 \in Ev_1^{obs} \setminus If_1\} \cup \\
& \{(2, e_2) \mid e_2 \in Ev_2^{obs} \setminus If_2\} \cup \\
& \{(e_1, e_2) \mid e_1 \in Ev_1^{obs} \wedge e_2 \in Ev_2^{obs} \wedge (e_1 \parallel e_2 \vee e_2 \parallel e_1)\}
\end{aligned}$$

$$getObs(e) = \begin{cases} (1, getObs_1(e_1)) & \text{if } e = (1, e_1) \\ (2, getObs_2(e_2)) & \text{if } e = (2, e_2) \\ getObs_1(e_1) *_O getObs_2(e_2) & \text{if } e = (e_1, e_2) \end{cases}$$

$$\begin{aligned}
Ev^{sec} = {} & \{(1, e_1) \mid e_1 \in Ev_1^{sec} \setminus If_1\} \cup \\
& \{(2, e_2) \mid e_2 \in Ev_2^{sec} \setminus If_2\} \cup \\
& \{(e_1, e_2) \mid e_1 \in Ev_1^{sec} \wedge e_2 \in Ev_2^{sec} \wedge (e_1 \parallel e_2 \vee e_2 \parallel e_1)\}
\end{aligned}$$

$$getSec(e) = \begin{cases} (1, getSec_1(e_1)) & \text{if } e = (1, e_1) \\ (2, getSec_2(e_2)) & \text{if } e = (2, e_2) \\ getSec_1(e_1) *_S getSec_2(e_2) & \text{if } e = (e_1, e_2) \end{cases}$$

For the examples in this chapter, we simply merge observations and secrets by pair building, i.e. $o_1 *_O o_2 = (o_1, o_2)$ and $s_1 *_S s_2 = (s_1, s_2)$. In this case, we will denote the view composition as $\mathcal{V}_1 \parallel \mathcal{V}_2$.

Before we present a composition operator for declassification bounds, let us discuss a (counter-)example that demonstrates one particular aspect that needs to be taken into account when composing bounds: the fact that the *scheduling* of secrets of the two components might depend on the scheduling of observations.

3.1.1. A Counterexample for Compositionality

Consider again the hiring workflow from Section 2.3.2, in particular the two medical examination processes Med_1 and Med_2. Let us denote their binary composition as $Meds$. Note that these two systems do not share any channels and therefore do not communicate with each other. They only communicate with their environment, receiving inputs on the channels $examine_i$, $report_i$, and $finish_i$, and providing output on the channels $stmt_i$, respectively. This special case of composition without synchronization is known as a product composition.

Recall that each of those components declassifies at most the equivalence class of the final version about the content of its medical reports (or the absence of a report). Hence, it is reasonable to expect that the composed system $Meds$ declassifies *both* equivalence classes of the reports (or their absence). Apart from that, we might still expect $Meds$ to keep the concrete sequences of content updates confidential. However, it turns out that observers may be able to deduce a bit of information about the *scheduling* of updates to the secret reports.

In order to see this, let us attempt to formalize a candidate declassification bound for $Meds$. Recall that, in general, such a bound shall specify a lower bound on the uncertainty of the observer: which secret sequences must be indistinguishable from which other secret sequences. In the case of $Meds$, we use the view $\mathcal{V}_{Med_1} \parallel \mathcal{V}_{Med_2}$, and the secret sequences are interleavings of secret sequences of Med_1 and Med_2, respectively. Hence, we combine the bounds of Med_1 and Med_2 by interleaving the pairs of secret sequences they contain. Formally, we define the composed bound to include sequences of secrets such that the projection to each component is included in the bound of that component:

$$B_{Meds} = \{(sl, sl') \in (Ev^{sec}_{Meds})^* \times (Ev^{sec}_{Meds})^* \mid (sl \restriction 1, sl' \restriction 1) \in B_{Med_1} \wedge (sl \restriction 2, sl' \restriction 2) \in B_{Med_2}\}$$

where B_{Med_1} and B_{Med_2} are copies of B_{Med}, where the channels in secret events are tagged with 1 and 2, respectively. This is the strongest bound we can hope to get for the composed system. It says that a secret sequence sl can be replaced by an arbitrary interleaving of the alternative secret sequences supported by the components. Since those declassify at most the equivalence class of the final report, the composed bounds declassifies at most the pair of equivalence classes.

Formally, recall that the bound B_{Med} presented in Section 2.2.5 requires the replacement of a (non-empty) sequence of secrets with an arbitrary other one, provided that the final reports are equivalent. For example, let us represent concrete medical reports by natural numbers for simplicity, where even numbers represent reports with

a positive outcome, and odd numbers represent reports with a negative outcome. For readability, we abbreviate channel names to their first letter in the following examples, and omit some of the redundant tagging of events. In this setting, the following pairs of sequences are contained in B_{Med_1} and B_{Med_2}, respectively:

$$(sl_1, sl_1') = (\langle r_1?1, r_1?4\rangle, \langle r_1?8\rangle) \in B_{Med_1}$$
$$(sl_2, sl_2') = (\langle r_2?3\rangle, \langle r_2?6, r_2?5\rangle) \in B_{Med_2}$$

Consequently, the following pair is an interleaving of (sl_1, sl_1') and (sl_2, sl_2') and contained in B_{Meds}:

$$(\langle r_1?1, r_1?4, r_2?3\rangle, \langle r_1?8, r_2?6, r_2?5\rangle) \in B_{Meds}$$

And this pair is unproblematic. We can apply the security properties of the components, obtain alternative component traces, and more or less concatenate them to obtain a composed alternative trace (with possibly just a bit of overlap in the observable events, if the second examination process began before the first one finished in the original trace). However, the following pair of secret sequences is *also* a valid interleaving contained in B_{Meds}:

$$(\langle r_1?1, r_1?4, r_2?3\rangle, \langle r_2?6, r_2?5, r_1?8\rangle) \in B_{Meds}$$

The only difference is that the ordering of the creation of the reports is reversed in the alternative sequence. Note that, on the left hand side (let us call this one sl), updates to report 1 come before those to report 2, while on the right hand side (let us call that one sl'), the second report comes first. Consider a trace $t \in [\![Meds]\!]$ producing sl, where it is observable that the process Med_1 finishes before the process Med_2 starts. It is impossible to construct a trace $t' \in [\![Meds]\!]$ producing sl', where Med_1 comes before Med_2 (as required by the observations), but report 1 is edited after report 2 (as required by sl'). Hence, the system *Meds* is *insecure* w.r.t. $\mathcal{V}_{Meds}$ and B_{Meds}.

This is somewhat surprising: Since the two systems do not communicate with each other at all, we might expect compositionality to hold trivially. Still, a naive attempt to compose the security properties of the two systems fails. The information that is leaked here is about the *scheduling* of the secret events of the two components. This information does not exist in the isolated components and only emerges during composition. From the scheduling of observable events in traces of *Meds*, it is possible for observers to deduce information about the scheduling of secret events. This is an actual, albeit subtle, information flow. However, in the example scenario, the scheduling of the medical reports is actually not confidential. Hence, we somehow have to specify that this information flow is allowed, in order for the composed system to satisfy BD Security.

3.1.2. Extending Views

One idea to solve this problem is to relax the desired security property of the composed system, declassifying information emerging during composition. An ad-hoc

repair for the example is to carefully restrict the composed declassification bound so that the scheduling of secrets is *preserved* from the original to the alternative sequence: only performing in-place substitution of secrets, and deletion or addition only directly next to secrets of the same component. However, it seems hard to generalize this idea into a robust and reusable approach, in particular if the dependencies of secrets on observations get more complex.

A more systematic approach is to change the *notion of secret* for the composed system, entirely removing any information about scheduling: We can define a secret producing function S that maps a trace not to an interleaving of secret sequences, but to the *pair* of sequences of secrets belonging to the two components, so that, for example, $S(t) = (sl_1, sl_2)$. This way, the fact that the scheduling is non-confidential is encoded into the structure of composed secrets. This corresponds to the intuitive idea that, if the components declassify the equivalence class of their secrets, then the composed system declassifies the pair of equivalence classes. The advantage of this approach is that compositionality then indeed holds trivially. The disadvantage is that the security property of the composed system has a different shape than that of the components, which complicates the further composition with additional components. Nevertheless, the flexibility of BD Security allows for this approach, and we explore this idea in more detail in Section 4.3.3.

In the main part of this chapter, we follow a different approach. The basic idea is to make the security properties of the *components* more precise, in order to specify the declassification of scheduling information explicitly. Note that, in each of the components, some information about the timing of secret inputs is known to observers already: when observers see an event on the $stmt_i$ channel, then they know that no more secret inputs on the $report_i$ channel will follow. However, this declassification is not encoded explicitly in the bounds; BD Security does not require us to encode it, because this is about a dependency of secrets on observations, whereas BD Security is mostly concerned with (a lack of) dependency of observations on secrets. This imprecision is unproblematic for the individual components, because we can still guarantee the possibility of all secret sequences as specified by the bound without interfering with the possibility of observations. For the composed system, however, the naive attempt at defining the bound claims that there is no dependency of the scheduling of secrets on the observations, and the system fails to guarantee this. In order to remedy this, we increase the precision of the component bounds by

- changing the notion of secrets of the components to include relevant information about the observations, and
- explicitly specifying the dependency of secrets on observations in the declassification bounds.

This allows us to specify the bound for the composed system so that the dependency of the scheduling of secrets on the scheduling of observations is sufficiently captured.

The example systems only produce secrets *after* an (observable) event on the *examine* channel, and *before* an event on the *stmt* channel. Hence, we add these

events to the set of secret events and obtain a new local view $\mathcal{V}'_{Med}$ by replacing Ev^{sec}_{Med} with

$$Ev^{sec'}_{Med} = Ev^{sec}_{Med} \cup (examine?\bullet) \cup (stmt!\bullet)$$

We specify the dependency of secrets on observations in the new local bound

$$\begin{aligned} B'_{Med} = \{(sl, sl') \in (Ev^{sec'}_{Med})^* \times (Ev^{sec'}_{Med})^* \mid (\exists (sl_0, sl'_0) \in B_{Med}, c, d. \\ sl \leq examine?c \cdot sl_0 \cdot stmt!d \wedge \\ sl' \leq examine?c \cdot sl'_0 \cdot stmt!d \wedge \\ sl =_{Ev^{obs}_{Med}} sl')\} \end{aligned}$$

It extends secret sequences taken from the original bound B_{Med} by prepending an *examine* event and appending a *stmt* event. We include not only "complete" sequences constructed this way, but also prefixes of such sequences; the condition $sl =_{Ev^{obs}_{Med}} sl'$ ensures that sl' begins with $examine?c$ and ends with $stmt!d$ if and only if sl does.

The system *Med* is still secure w.r.t. the extended view $\mathcal{V}'_{Med}$ and bound B'_{Med}. This is not surprising, since the extended property only makes information explicit that is already known statically from the system specification.

3.1.3. Composing Bounds

We combine two copies of B'_{Med} to the composed bound B'_{Meds} as above, with the difference that we now have to take into account the preservation of observations, as well:

$$\begin{aligned} B'_{Meds} = \{(sl, sl') \in (Ev^{sec'}_{Meds})^* \times (Ev^{sec'}_{Meds})^* \mid (sl \upharpoonright 1, sl' \upharpoonright 1) \in r_1(B'_{Med}) \wedge \\ (sl \upharpoonright 2, sl' \upharpoonright 2) \in r_2(B'_{Med}) \wedge \\ sl =_{Ev^{obs'}_{Meds}} sl'\} \end{aligned}$$

When moving from an original sequence of secrets sl to an alternative sequence sl', the additional condition $sl =_{Ev^{obs'}_{Meds}} sl'$ ensures that, in particular, the *scheduling* of the observations included in $Ev^{obs'}_{Med} \cap Ev^{sec'}_{Med}$ is preserved. Furthermore, the extended local bounds ensure that the compatibility of secrets and observations is preserved. Combined, the compatibility of the schedulings of observations and secrets is now guaranteed to be preserved.

For example, consider the problematic pair discussed above:

$$(sl, sl') = (\langle r_1?1, r_1?4, r_2?3\rangle, \langle r_2?6, r_2?5, r_1?8\rangle) \in B_{Meds}$$

The following pair is an extension of (sl, sl'), inserting *examine* and *stmt* events as specified by B'_{Med}:

$$\begin{aligned} (sl'', sl''') = (\, \langle e_1?c, r_1?1, r_1?4, s_1!\text{True}, e_2?c, r_2?3, s_2!\text{False}\rangle, \\ \langle e_2?c, r_2?6, r_2?5, s_2!\text{False}, e_1?c, r_1?8, s_1!\text{True}\rangle) \end{aligned}$$

This pair is *not* included in B'_{Meds} any more, due to the difference in observable events. Hence, the problem discussed above does not occur any more. It turns out that the extended security properties of the components are indeed sufficiently strong for compositionality, and therefore *Meds* satisfies BD Security w.r.t. $\mathcal{V}'_{Meds} = \mathcal{V}'_{Med_1} \parallel \mathcal{V}'_{Med_2}$ and B'_{Meds}. This follows from an abstract compositionality result, presented in the next section.

3.2. Compositionality of BD Security in Abstract Terms

Our abstract compositionality result will place constraints on the declassification bound it allows to be derived for the composed system, and the possible behaviors that the subsystems must exhibit in order for the result to be applicable. We will formulate these constraints in terms of two notions called observation-secret compatibility and well-behavedness. We use the former to capture the previous section's idea of ensuring that the scheduling constraints induced by secrets and observations, respectively, do not contradict each other. The latter will help us prove BD Security of the composed system by invoking the security guarantees of the components.

3.2.1. An Abstract Compositionality Result

In order to explain those notions, let us step back for a moment and return to the most abstract formulation of BD Security (cf. Section 2.2.2), leaving underspecified what system behaviors, observations, and secrets are, and focusing purely on the *relations* between them. The advantage is that, on this level of abstraction, we can succinctly capture the essence of the concepts of well-behavedness and (preservation of) observation-secret compatibility, and how they help proving compositionality. After this subsection, the rest of this chapter will elaborate the instantiation of these notions for the composition of LTSs as introduced in Chapter 2.

The only assumption about the structure of systems that we make in this subsection is that they can be characterized by their set of possible behaviors. In the rest of this thesis, we focus on finite traces of events, but the abstract compositionality result presented in this subsection is agnostic to the choice of universe of behaviors, and could also be instantiated for other representations such as infinite streams or event structures [Win87]. Given a universe of behaviors Beh, a system *Sys* is characterized by the subset $[\![Sys]\!] \subseteq \mathsf{Beh}$ of behaviors that the system might actually exhibit.

In this subsection, we consider composition abstractly w.r.t. some arbitrary but fixed composition operator $\oplus\colon \mathsf{Beh}_1 \times \mathsf{Beh}_2 \to 2^{\mathsf{Beh}}$, which takes two subsystem behaviors (drawn from the universes of behaviors Beh_1 and Beh_2, respectively) and returns a set of possible compositions of those behaviors. This induces a notion of composition of systems: A behavior is possible in the composed system iff it is a composition of two possible subsystem behaviors. Formally, a composition of Sys_1

and Sys_2 via $\oplus$ is a system Sys such that

$$[\![Sys]\!] = \{t \mid \exists t_1 \in [\![Sys_1]\!], t_2 \in [\![Sys_2]\!].\ t \in t_1 \oplus t_2\}$$

The composition of LTSs we introduced in Chapter 2 is an instance of this, with traces (Ev^*) as behaviors and synchronized interleaving ($\|$) as the composition operator.

Let Sys be a composition of Sys_1 and Sys_2. We consider BD Security of these systems (in the abstract formulation of Definition 2.1) with respect to some (arbitrary but fixed) observation and secret extraction functions and bounds

$$\begin{array}{lll} O\colon \mathsf{Beh} \to \mathsf{O} & O_1\colon \mathsf{Beh}_1 \to \mathsf{O}_1 & O_2\colon \mathsf{Beh}_2 \to \mathsf{O}_2 \\ S\colon \mathsf{Beh} \to \mathsf{S} & S_1\colon \mathsf{Beh}_1 \to \mathsf{S}_1 & S_2\colon \mathsf{Beh}_2 \to \mathsf{S}_2 \\ B \subseteq \mathsf{S} \times \mathsf{S} & B_1 \subseteq \mathsf{S}_1 \times \mathsf{S}_1 & B_2 \subseteq \mathsf{S}_2 \times \mathsf{S}_2 \end{array}$$

We want to derive this security property of Sys from the security properties of Sys_1 and Sys_2. For brevity, we collect these parameters in a tuple

$$\mathcal{CS} = (O_1, S_1, \oplus, O_2, S_2, O, S)$$

called the *composition setup*. Note that we do not assume here that the parameters O, S, and B are constructed from those of the subsystems in a specific way. Nevertheless, in order to formulate conditions for compositionality, we need to establish *some* connection between those parameters. We base this connection on the given composition operator for system behaviors, and derive from it composition operators for observations and secrets (and, on top of that, for declassification bounds further below). We consider a secret s to be a composition of two component secrets s_1 and s_2 iff s is the secret of *some* behavior that is a composition of behaviors with secrets s_1 and s_2, respectively (and analogously for observations).

Definition 3.2. The composition operators for observations and secrets w.r.t. $\mathcal{CS}$, denoted $\oplus_O$ and $\oplus_S$, respectively,[1] are defined as

$$\begin{aligned} o_1 \oplus_O o_2 &= \{O(t) \mid \exists t_1 \in \mathsf{Beh}_1, t_2 \in \mathsf{Beh}_2.\ t \in t_1 \oplus t_2 \wedge O_1(t_1) = o_1 \wedge O_2(t_2) = o_2\} \\ s_1 \oplus_S s_2 &= \{S(t) \mid \exists t_1 \in \mathsf{Beh}_1, t_2 \in \mathsf{Beh}_2.\ t \in t_1 \oplus t_2 \wedge S_1(t_1) = s_1 \wedge S_2(t_2) = s_2\} \end{aligned}$$

In particular, $t \in t_1 \oplus t_2$ implies $O(t) \in O_1(t_1) \oplus_O O_2(t_2)$ and $S(t) \in S_1(t_1) \oplus_S S_2(t_2)$. We will use these operators to formulate constraints on the declassification bound we can guarantee for the composed system and the behaviors that must be possible in the subsystems.

Regarding the declassification bound, the composed system intuitively declassifies *at least* as much information as the components *combined*. Additional information may be declassified due to scheduling, as we have seen above, or due to synchronization of the components. The latter did not occur in the above composition of Med_1

[1] Formally, these operators also need to be indexed by $\mathcal{CS}$. For brevity, we instead make the composition setup clear in the context and omit the indices.

and Med_2, since those systems do not communicate with each other, but in general, the synchronization that occurs upon communication can reveal additional aspects of secret information to an observer. For example, consider a composition of two systems *Div* and *Sgn*, depicted in Figure 3.1, where the first receives an integer i as (local) input from a user and sends $i \div 2$ to the second component (where $\div$ denotes integer division). The second system outputs the *sign* of $i \div 2$. Assuming that the output channel of *Sgn* is observable, while the input of *Div* as well as the channel between *Div* and *Sgn* are secret, and assuming that we consider this composed system to be secure, then the system declassifies that

- the output of *Div* corresponds to its input divided by 2,
- the output of *Sgn* corresponds to the sign of its input, and
- *combined*, the sign of the original secret input is declassified in the observable output.

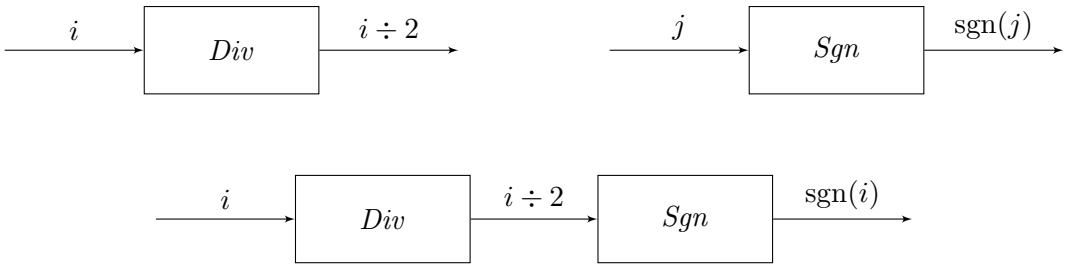

Figure 3.1.: Composing integer division and sign extraction

This suggests that the amount of information declassified by the composed system corresponds to a kind of "synchronized union" of the information declassified by the components. Formally, however, recall that declassification bounds do not directly specify the amount of declassified information, but are defined the other way around: they specify a *lower bound* on the *uncertainty* of the observers about the secret, and synchronization can decrease uncertainty. Hence, we define a composition operator for declassification bounds corresponding to a "synchronized intersection", where a pair (s, s') of composed secrets is included in the composed bound only if we are guaranteed to find a decomposition into pairs of local secrets such that *both* are included in the bounds of the components, i.e., if

$$\forall s_1, s_2.\ s \in s_1 \oplus_S s_2 \longrightarrow (\exists s_1', s_2'.\ s' \in s_1' \oplus_S s_2' \ \wedge (s_1, s_1') \in B_1 \wedge (s_2, s_2') \in B_2)$$

For example, consider again the composition of *Div* and *Sgn*. Due to the synchronization of secrets, the message sent by *Div* is the one received by *Sgn*. Let the bound B_1 for *Div* require that the user input i can be replaced by any i' (provided that the message sent is replaced by $i' \div 2$ accordingly). Let the bound B_2 for *Sgn*

require arbitrary replacement of the received message, provided that the new value has the same sign as the old. The composition merges these bounds and requires the replacement of i not with arbitrary integers, as B_1 requires, but only with integers of the same sign, due to B_2.[2]

Another aspect we have to consider when composing bounds is the relative scheduling of secrets and observations. Recall that, in the example composing $Meds_1$ and $Meds_2$, we had the problem of contradictory constraints arising from the alternative secret sequence and the observation sequence, respectively. We solved this problem by enriching the secret sequences with relevant observable events, recording the dependency of secrets on observations in the component bounds, and constraining the composed bound to make the preservation of the scheduling of observable events explicit. We lift the latter to the abstract setting of this subsection using the notion of observation-secret compatibility relation: We consider an observation o and a secret s to be compatible if it is possible to construct *some* behavior that exhibits *both* o and s. In the case of *Meds*, the following is a valid observation-secret compatibility relation:

$$\left\{(ol, sl) \mid ol =_{Ev^{obs'}_{Meds} \cap Ev^{sec'}_{Meds}} sl\right\}$$

Such a compatibility relation C also induces a preorder on just the *secrets* w.r.t. the sets of observations they are compatible with, denoted $\preceq_C$. For *Meds*, this is actually an equivalence relation, where two sequences of secrets events are equivalent if they contain the same sequence of observable events:

$$sl \preceq_{C_{Meds}} sl' \longleftrightarrow sl =_{Ev^{obs'}_{Meds}} sl'$$

In general, we define these notions as follows.

Definition 3.3. An *observation-secret compatibility relation between O and S* is a relation $C \subseteq \mathsf{O} \times \mathsf{S}$ such that, for all $t \in \mathsf{Beh}$, it holds that $(O(t), S(t)) \in C$. We say that *o is C-compatible with s* if $(o, s) \in C$.

The observations compatible with s w.r.t. C are *covered by s'*, denoted $s \preceq_C s'$, iff all $o \in \mathsf{O}$ that are C-compatible with s are also C-compatible with s'.

We use the relation $\preceq_C$ to constrain the declassification bound for the composed system. We only require the possibility of replacing s by s' if the latter is compatible with *at least* the observations that the former is compatible with. This leads us to the following definition of the bound composition operator $\oplus^C_B$ (where we again omit the index $\mathcal{CS}$ for readability).

Definition 3.4. Let C be an observation-secret compatibility relation between O and S. The *composition operator for declassification bounds w.r.t. $\mathcal{CS}$ and C*, denoted $\oplus^C_B$, is defined as

[2] More precisely, it requires that the secret user input i can be replaced by i' and the message $i \div 2$ by $i' \div 2$, as required by B_1, provided that $i' \div 2$ has the same sign as $i \div 2$, as required by B_2. The latter is equivalent to the fact that i and i' themselves have the same sign.

$$B_1 \oplus_B^C B_2 = \{(s, s') \mid \forall s_1, s_2.\ s \in s_1 \oplus_S s_2 \longrightarrow (\exists s_1', s_2'.\ s' \in s_1' \oplus_S s_2' \wedge (s_1, s_1') \in B_1 \wedge (s_2, s_2') \in B_2 \wedge s \preceq_C s')\}$$

This generalizes the construction of B'_{Meds} from B'_{Med_1} and B'_{Med_2} from page 32. Note that the constraint $s \preceq_C sl'$ makes the bound smaller and therefore might leak information, depending on the definitions of O, S, and C. However, the compatibility relations we will focus on only state that observable information is preserved (see $\preceq_{C_{Meds}}$, for example), so they do not leak any additional information about the secrets. Also note that Definition 3.3 allows us to choose a C that is *bigger* than $\{(O(t), S(t)) \mid t \in \mathsf{Beh}\}$. For example, we could choose the total relation, which means that $\preceq_C$ also becomes the total relation, simplifying the composition of bounds. This might be useful if we know that the scheduling of observations and secrets cannot pose any problems, for example, if only one of the components generates secret information. We will discuss such a scenario in Chapter 4.

In order to identify what is still missing for compositionality, let us discuss the proof idea. Given a trace $t \in [\![Sys]\!]$ and a secret s' such that $(S(t), s') \in B_1 \oplus_B^C B_2$, we have to show the existence of a possible system behavior $t' \in [\![Sys]\!]$ with the observation $O(t)$ and the secret s'. Since Sys is a composition of Sys_1 and Sys_2 via $\oplus$, we obtain $t_1 \in [\![Sys_1]\!]$ and $t_2 \in [\![Sys_2]\!]$ with $t \in t_1 \oplus t_2$. By the definition of $B_1 \oplus_B^C B_2$, we can decompose s' into s_1' and s_2' such that $(S_i(t_i), s_i') \in B_i$. This allows us to invoke the security properties of the components, obtaining t_i' with $O_i(t_i') = O_i(t_i)$ and $S_i(t_i') = s_i'$. Now we just have to merge these alternative component traces back into the desired trace $t' \in [\![Sys]\!]$ with $O(t') = O(t)$ and $S(t') = s'$.

However, this is not possible unconditionally, as we have seen for the system *Meds* in Section 3.1.1, where the merging failed due to incompatible scheduling in $O(t)$ and s'. Hence, we restrict our attention to alternative secrets s' that are *compatible* with the original observation. We call a composed system *well-behaved* if the merging of component traces is guaranteed to be possible, provided that their observations and secrets synchronize to *compatible* observations and secrets of the composed system.

Definition 3.5. *Sys* is a *well-behaved composition of* Sys_1 *and* Sys_2 *w.r.t.* $\mathcal{CS}$ iff for all $o \in \mathsf{O}$, $s' \in \mathsf{S}$, $t_1' \in [\![Sys_1]\!]$, and $t_2' \in [\![Sys_2]\!]$ such that o is C-compatible with s', $o \in O_1(t_1') \oplus_O O_2(t_2')$, and $s' \in S_1(t_1') \oplus_S S_2(t_2')$, there exists $t' \in [\![Sys]\!]$ such that $S(t') = s'$ and $O(t') = o$.

This condition is sufficient for compositionality of BD Security.

Theorem 3.1. *Let Sys be a well-behaved composition of* Sys_1 *and* Sys_2 *w.r.t.* $\mathcal{CS}$.

If each Sys_i *satisfies BD security w.r.t.* O_i, S_i, *and* B_i, *then Sys satisfies BD security w.r.t.* O, S, *and any* B *with* $B \subseteq B_1 \oplus_B^C B_2$.

With the assumption of well-behavedness, the above proof attempt can be finished as follows. We know from Definition 3.3 that $O(t)$ and $S(t)$ are C-compatible. In order to *preserve* this compatibility, we included the condition involving $\preceq_C$ in Definition 3.4. With $(S(t), s') \in B \subseteq B_1 \oplus_B B_2$, it allows us to derive $S(t) \preceq_C s'$. Hence, $O(t)$ and s' are C-compatible, too. This allows us to invoke well-behavedness to merge t'_1 and t'_2 and obtain the desired trace $t' \in [\![Sys]\!]$.

Proving well-behavedness is the main challenge when composing BD Security properties. The rest of this chapter is devoted to exploring how we can guarantee well-behavedness for compositions of labeled transition systems.

3.2.2. An Example for Well-Behavedness

Consider again the system *Meds*. The composition setup consists of the observation and secret producing functions corresponding to $r_1(\mathcal{V}'_{Med})$, $r_2(\mathcal{V}'_{Med})$, and $\mathcal{V}'_{Meds}$, respectively, together with the compatibility relation C_{Meds} and the trace composition operator $\parallel$ between Med_1 and Med_2.

Since, in all views in this example, the observations and secrets consists of the events themselves (the functions *getObs* and *getSec* are the identity), the composition operators $\|_O$ and $\|_S$ are identical to the trace composition operator $\parallel$ (restricted to sequences of observable or secret events, respectively). This also implies that $r_1(B'_{Med}) \|_B r_2(B'_{Med})$ is indeed equal to B'_{Meds}: The latter respects C_{Meds} by definition. Furthermore, the quantified formula in Definition 3.4 involving the trace composition operator can be simplified to the projection-based definition of B'_{Meds}, due to the fact that the decomposition of secrets is unique for $\mathcal{V}'_{Meds}$.[3]

In order apply Theorem 3.1, we have to prove well-behavedness, i.e. construct a composed trace t' for given t'_1, t'_2, ol and sl'. Since the systems do not communicate with each other, we just have to find a suitable interleaving of t'_1 and t'_2. This is trivial: Since the secret sequence sl' contains an interleaving of all observable and secret events of t'_1 and t'_2 (recall that $Ev^{obs'}_{Meds} \subseteq Ev^{sec'}_{Meds}$), it is almost the desired trace t' already; we just have to insert the non-observable and non-confidential events at suitable positions. In this case, these are the events on the $finish_i$ channels, which we insert at the same positions they occur in the traces t'_i: directly before the $stmt_i$ events. We now have a suitable trace t' of *Meds*, producing the secret sequence sl'. It remains to show that it also produces the observation sequence ol. From the assumption $(ol, sl') \in C_{Meds}$, we get $ol =_{Ev^{obs'}_{Meds} \cap Ev^{sec'}_{Meds}} sl'$. With $S_{\mathcal{V}_{Meds}}(t') = sl'$ and $Ev^{obs'}_{Meds} \subseteq Ev^{sec'}_{Meds}$, this implies $O_{\mathcal{V}_{Meds}}(t') = ol$.

This direct proof of well-behavedness is straightforward, but it seems more tedious than necessary, especially given the fact that the composition setup is so simple. Indeed, the well-behavedness of *Meds* follows trivially from Lemma 3.3, which we will present in Section 3.4. This general result will have to address a few more challenges, demonstrated by the following counterexample.

[3] Since secrets of *Meds* are traces of (composite) events, and the decomposition of traces is unique: $t \in t_1 \parallel t_2$ implies $t_1 = t \restriction 1$ and $t_2 = t \restriction 2$.

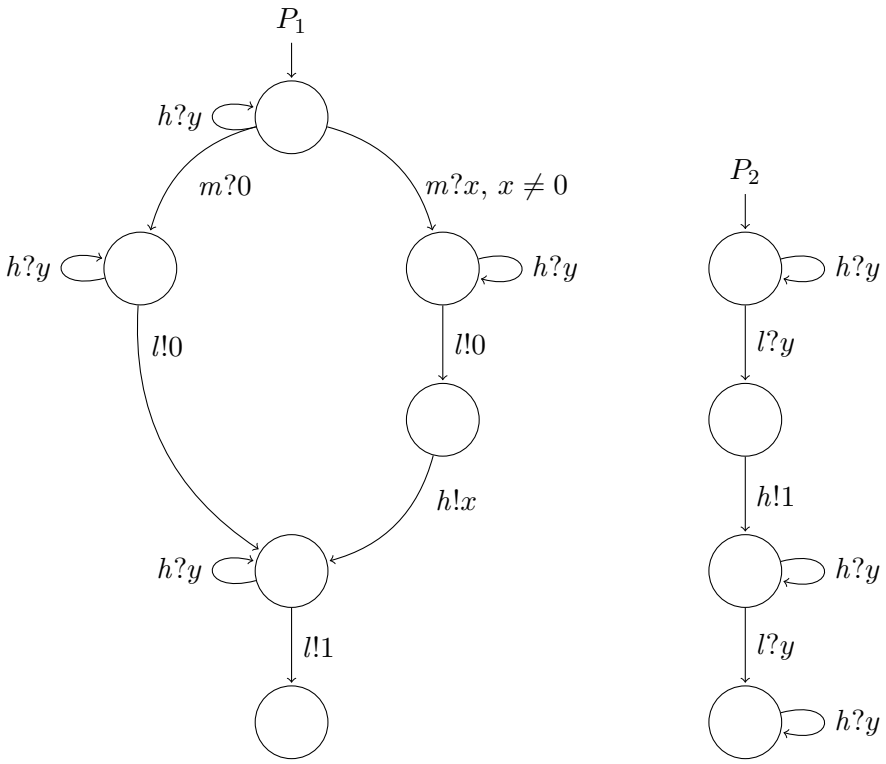

Figure 3.2.: PSNI example processes

3.2.3. A Counterexample for Well-Behavedness

We now show that the simple, PSNI-like property that we sketched in Section 2.2.4 is *not* compositional (although the original formulation of PSNI in [RS14] is, and we will present a compositional version of our PSNI-like property in Section 3.4.2). Consider the systems P_1 and P_2, depicted in Figure 3.2. The process P_1 receives a confidential input on an encrypted channel, and outputs first 0, then 1 on the low channel. In case the encrypted input x was non-zero, it also sends x on the h channel directly after $l!0$. P_2, on the other hand, *receives* two messages on the low channels; in between, it sends 1 on the h channel. Both processes can receive arbitrary input on the h channel, respectively, except directly before they produce high output.

Hence, both of these systems satisfy the PSNI-like property described in Section 2.2.4 w.r.t. $L = \{l\}$, $M = \{m\}$, $H = \{h\}$, because they allow the insertion and deletion of inputs on h without interfering with observations on l, and the arbitrary replacement of inputs on m (although only P_1 actually communicates on m with the environment).

Let us attempt to prove well-behavedness of the composition of P_1 and P_2. Consider the two traces

$$t_1 = \langle m?1, l!0, h!1, h?1, l!1 \rangle$$
$$t_2 = \langle l?0, h!1, h?1, l?1 \rangle$$

of P_1 and P_2, respectively. Let $ol = \langle m?\mathtt{d}, l.0, l.1 \rangle$ (where $c.m$ denotes a synchronization between $c!m$ and $c?m$) and $sl' = \langle m?1 \rangle$. The two sequences ol and sl' are compatible in the sense that they agree on the occurrence of events in the intersection of observable and secrets events, namely, $m?1$. Moreover, it holds that the observations and secrets of t_1 and t_2 compose to ol and sl', respectively:

$$ol \in \langle m?\mathtt{d}, l!0, l!1 \rangle \parallel_O \langle l?0, l?1 \rangle$$
$$sl' \in \langle m?1, h?1 \rangle \parallel_S \langle h?1 \rangle$$

Witnesses for this are t_1 and the trace $t_2' = \langle l?0, h?1, h!1, l?1 \rangle$, which has the same observations and secrets as t_2 and composes with t_1 to a trace

$$t' = \langle m?1, l.0, h.1, h.1, l.1 \rangle$$

that produces ol and sl'. Note that high outputs (and synchronization events including high outputs) are not considered confidential w.r.t. PSNI and therefore do not appear in secret sequences.

However, t' is *not* a trace of the composition of P_1 and P_2, because the ordering of communication events on the channel h does not match. Neither is it possible to find other suitable traces of P_1 and P_2 that synchronize: Indeed, after the first synchronization on l, P_2 *always* insists on sending on h before receiving on h, while P_1 also insists on sending on h before receiving on h if something else than 0 has been received on m. Hence, the process gets stuck in a deadlock after receiving $m?x$ with $x \neq 0$ and synchronizing on l: Both processes try to send, but neither of them is ready to receive. The composed process can therefore be simplified to

$$P_1 \parallel P_2 = m?x \to l.0 \to (\textbf{if } x = 0 \textbf{ then } (h.1 \to l.1))$$

This process is *insecure* w.r.t. any PSNI property with $m \in M$ and $l \in L$: If the final $l.1$ occurs, an observer knows that the secret input on m must have been 0.

The problem is different from the one we discussed above in the context of *Meds*. It is not about the relative scheduling of observations and secrets, but about the synchronization of secret events with ones that are *neither* observable nor secret; in this case, the high outputs. We call these events *neutral*.

It is possible to solve this problem again by strengthening the views; not by adding observable events to the secrets, but by making neutral interface events secret. However, this would require us to specify the dependency of high outputs on the inputs in the declassification bound. Hence, knowledge about the system specification is required to formulate the security property. For an intentionally generic security property such as PSNI, this is undesirable; it should be applicable to arbitrary systems. In the rest of this chapter, we follow a different approach, inspired by the compositionality results of the MAKS framework: We strengthen the notion of BD Security itself.

3.3. Challenges for the Well-Behavedness of LTSs

When proving well-behavedness, we are given two component traces and (compatible) compositions of their observations and secrets, and we have to prove the existence of a composed trace producing those observations and secrets. As demonstrated by the examples we have discussed so far, there are several challenges we need to overcome for a general compositionality result. In this section, we deep dive into each of those challenges, and identify a set of sufficient side conditions to address them.

As another example, consider the situation in Figure 3.3. It depicts two component traces; let us call the upper one t_1, and the lower one t_2. Let us assume that the two components communicate on the observable channel l, the encrypted channel m (where message occurrence is observable, but not content), and the secret channel h. The first component sends on l and m and receives on h, and vice versa for second component. Moreover, the components receive input from the environment on channels h_1 and h_2, respectively. Let us further assume that the components satisfy the PSNI-like security property of Section 2.2.4 with $L_i = \{l\}$, $M_i = \{m\}$, and $H_i = \{h, h_i\}$, and that we want to derive this property for the composition of the systems as well.

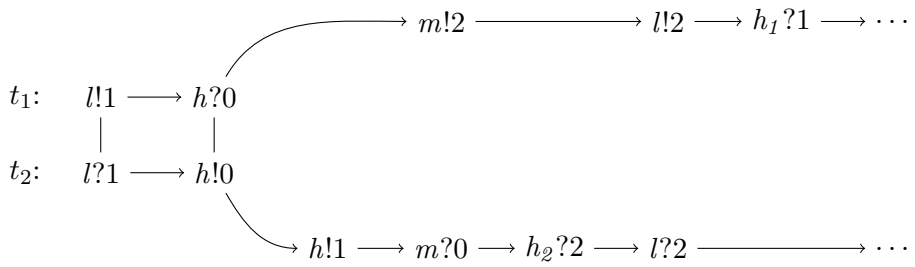

Figure 3.3.: Merging example

The example traces t_1 and t_2 in Figure 3.3 do not synchronize; in particular, the output event $h!1$ in t_2 is not matched by an input event in t_1. However, their observations and secrets can be composed, for example as follows:

$$ol = \langle l.1, m.\mathtt{d}, l.2\rangle \in \langle l!1, m!\mathtt{d}, l!2\rangle \parallel_O \langle l?1, m?\mathtt{d}, l?2\rangle$$
$$sl = \langle h_1?1, h_2?2\rangle \in \langle h?0, h_1?1\rangle \parallel_S \langle m?0, h_2?2\rangle$$

In order to see how $S_1(t_1)$ and $S_2(t_2)$ compose to sl, recall that output events (and synchronization events, which include outputs) are not considered confidential for PSNI, and therefore disappear in secret sequences. Witnesses for the composition are the following two sequences of events t_1' and t_2', which yield the same observations and secrets as t_1 and t_2, respectively, and can be synchronized to t', which yields ol and sl:

$$t'_1 = \langle l!1, h?0, h?1, m!2, l!2, h_1?1 \rangle$$
$$t'_2 = \langle l?1, h!0, h!1, m?2, l?2, h_2?2 \rangle$$
$$t' = \langle l.1, h.0, h.1, m.2, l.2, h_1?1, h_2?2 \rangle$$

Since the observations and secrets of t_1 and t_2 compose to ol and sl, well-behavedness requires us to construct a merged trace of the composed system with those observations and secrets. We know that there are *some* sequences of events t'_1 and t'_2 that would be suitable, but it is not given that they are actually *possible traces* of the given components. In order to guarantee the existence of such traces, we follow an approach based on what is known in the literature as a *zipping lemma*: We formulate additional requirements on the components so that, given a pair of traces t_1 and t_2 which have matching secrets and observations but do not compose, we can "repair" those traces by making local changes so that the adapted traces do synchronize. In general, we have to overcome the following challenges for the synchronization of traces:

1. Failure to synchronize with *neutral* events, for example the event $h!1$ in t_2 mentioned above. This problem also occurred in the PSNI example of Section 3.2.3.

2. Even for interface events that are not neutral, matching observations and secrets might not be sufficient to guarantee synchronization. Consider, for example, encrypted communication events on the channel m in Figure 3.3. The events $e_1 = m!2$ and $e_2 = m?0$ have matching observations and secrets (take as a witness the event $e'_1 = m!0$, which produces the same observation $m!\mathtt{d}$ as e_1, is not secret, just as e_1, and it synchronizes with e_2), but e_1 and e_2 themselves fail to synchronize.

3. Incompatible relative scheduling of observable and secret events, as we have seen in Section 3.1.1 concerning the example *Meds*. In Figure 3.3, note that in the traces t_1 and t_2, the event $h_2?2$ causally precedes $h_1?1$ due to the observable communication on l in between them, while the composed sequence of secrets sl requires us to produce $h_1?1$ first, followed by $h_2?2$.

Intuitively, these challenges can be traced back to an *asymmetry* of the security guarantees of the individual components. We may relax the security guarantee of one component, by choosing some interface events to be neutral, by abstracting away details of events in the observations and secrets, or by ignoring the scheduling of secret and observable events in the bound, but when we do that, we might have to strengthen the security guarantee of the *other* component for compositionality. We now discuss each of these challenges in more detail and propose strengthened BD Security properties that are sufficient to solve them.

3.3.1. Neutral events

The first challenge is that any of the two given component traces might contain communication events that are neutral (i.e., neither observable nor secret) that are not matched by a corresponding communication event in the other trace. This can happen even if the observations and secrets of the two traces can be composed, because neutral events do not appear in those. In order to solve this problem, we require it to be possible to insert communication events that match neutral events *at any position* in a trace, without interfering with observations and secrets. This guarantees that the occurrence of neutral communication events cannot compromise confidentiality. This requirement leads to a trade-off: On the one hand, declaring some communication events to be neutral for a component gives us more flexibility when proving the *local* security property of that component, since we can use neutral events freely when constructing alternative traces. On the other hand, when *composing* systems, we have to prove that this flexibility at the communication interface cannot compromise security, by placing this additional requirement on the *other* component. For example, for PSNI we chose high output events to be neutral, allowing each system to freely emit high outputs, but demanding that it accepts arbitrary high *inputs* without interfering with observations.

Concretely, in the example of Figure 3.3, we want to insert $h?1$ in t_1 before $m!2$ in order to proceed with the synchronization of t_1 and t_2. In the PSNI example of Section 3.2.3, we want to insert $h_1?1$ *before* instead of after $h_2!1$, allowing us to obtain an alternative trace of P_2 suitable for synchronization with the trace t_1 of P_1.

In order to formalize this idea, let

$$N_{LTS}^{\mathcal{V}} = Ev \setminus (Ev^{obs} \cup Ev^{sec})$$

denote the set of events of LTS that are neutral in $\mathcal{V}$. In the following definitions, we will consider the composition of systems LTS_1 and LTS_2 with views $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively, and will abbreviate $N_{LTS_i}^{\mathcal{V}_i}$ as N_i for readability. Applying this to the above example, we get $h!1 \in N_2$. We want to be able to insert the corresponding input event $h?1$ into t_1. More generally, we define the *communication complement* of a set of events N, i.e., events of the *other* component matching an event in N, as

$$\overline{N} = \{e' \mid \exists e \in N.\ e \parallel e' \vee e' \parallel e\}$$

For example, $h!1 \in N_2$ implies $h?1 \in \overline{N_2}$. When composing two systems LTS_1 and LTS_2 with neutral events at the communication interface, we require that LTS_1 accepts the events in $\overline{N_2}$ at any time without interfering with observations, and vice versa for LTS_2 and $\overline{N_1}$. These requirements can be expressed as BD Security properties, by choosing bounds B_i that support the arbitrary insertion and deletion of events in $\overline{N_j}$. We call such a bound B_i total in $\overline{N_j}$.

Definition 3.6. A declassification bound B is *total in* $Z \subseteq Ev^{sec}$ *for* $\mathcal{V}$ iff B is reflexive and for all $t, t' \in (Ev^{sec})^*$ with $O_{\mathcal{V}}(t) = O_{\mathcal{V}}(t')$, it holds that

$$(S_{\mathcal{V}}(t), S_{\mathcal{V}}(t')) \in B \longleftrightarrow (S_{\mathcal{V}}(t \setminus Z), S_{\mathcal{V}}(t' \setminus Z)) \in B$$

In the above example, BD Security w.r.t. a bound that is total in $\overline{N_2}$ allows us to insert $h?1$ into a trace, since $h?1 \in \overline{N_2}$. In particular, such a bound includes the pair $(\langle h?0, h_1?1\rangle, \langle h?0, h?1, h_1?1\rangle)$. Hence, it allows us to insert $h?1$ into t_1 after $h?0$. Note that this requires the events in $\overline{N_j}$ to be *secret* in $\mathcal{V}_i$; otherwise, we could not specify requirements about their insertion in the declassification bound. Hence, at most *one* event in a pair of corresponding input and output events may be neutral. For simplicity, we choose to allow only neutral *output* events. This is in line with the common assumption that a system controls the outputs it produces, but not the inputs it receives. The first part of the following definition formalizes that only outputs may be neutral by requiring that input events must be either observable or secret. Moreover, it requires that an output event is observable iff the corresponding input is. If an output event is neutral, then the input must be secret, while a non-neutral output is secret iff the input is secret. Finally, the second part of the following definition requires that the information contained in the observations and secrets of input events fully captures their synchronization behavior (including the fact whether they synchronize with neutral events). This guarantees that the secrets used in a bound that is total in $\overline{N_j}$ are actually precise enough to guarantee that we can insert suitable events that synchronize with a given event in N_j.

Definition 3.7. We say two views $\mathcal{V}_1$ and $\mathcal{V}_2$ are *composable w.r.t.* $\parallel$ iff

1. for all $e \in Out_i$ and $e' \in In_j$ with $e \parallel e'$
   - $e' \in Ev_j^{obs} \cup Ev_j^{sec}$
   - $e \in Ev_i^{obs} \longleftrightarrow e' \in Ev_j^{obs}$
   - $e \in Ev_i^{sec} \longrightarrow e' \in Ev_j^{sec}$ and $e' \in Ev_j^{sec} \cap Ev_j^{obs} \longrightarrow e \in Ev_i^{sec}$
2. for all $e \in Out_i$ and $e', e'' \in In_j$ with $e' \approx_{\mathcal{V}_j} e''$ (cf. Definition 2.3), it holds that $e \parallel e' \longleftrightarrow e \parallel e''$ and $e' \in \overline{N_i} \longleftrightarrow e'' \in \overline{N_i}$.

With these assumptions, a BD Security property w.r.t. a bound that is total in $\overline{N_j}$ guarantees that we can insert exactly the events that we need for synchronization with neutral events. However, it does not give us precise guarantees about *where* in the trace these events will appear. For example, in the trace t_1 of Figure 3.3, we want to insert $h?1$ not just somewhere after $h?0$, but *directly* after. Inserting it after $m!2$ or $l!2$, for example, would not help us in synchronizing t_1 with t_2. Hence, we additionally require that insertions of events in $\overline{N_j}$ are possible at *any position* in a trace. BD Security itself is too weak for capturing this requirement: it is only designed to control which sequences of secrets must be possible, but not the position of secret events relative to observable or neutral ones. Hence, we tailor BD Security by formulating variants of it that give us more control over the adaptations of traces that we make for synchronization.

In particular, we require these adaptations to be incremental in the sense that the resulting trace is *unchanged* up to the point of the local change. This will allow us to prove well-behavedness by applying a technique based on *zipping lemmas* (used,

for example, in the MAKS framework [Man02]): We make adaptations to the local traces from left to right, keeping the parts that are already merged intact, and "closing the zipper" one step at a time. For example, in Figure 3.3, the first two pairs of communication events synchronize already to $\langle l.1, h.0\rangle$ and should be kept intact when we proceed, for example, to insert $h?1$ between $h?0$ and $m!2$ in t_1.

For this purpose, we formulate a property that considers all prefixes β of system traces, i.e., it splits a trace t into $\beta \cdot \alpha$ for some suffix α. If the declassification bound requires a secret to be produced after β that can be produced by one of the events we want to insert, then the property requires the insertion of a corresponding event *immediately* after β.

Definition 3.8. Let $X \subseteq Ev_{\mathcal{V}}^{sec}$ be a set of secret events.

LTS supports *eager insertion of X for $\mathcal{V}$ and B*, abbreviated $EI[X, \mathcal{V}, B]$, iff for all $\beta, \alpha \in Ev^*$, $x \in X$, and $sl' \in (Ev_{\mathcal{V}}^{sec})^*$ such that

$$\beta \cdot \alpha \in [\![LTS]\!] \wedge \big(S_{\mathcal{V}}(\beta \cdot \alpha), S_{\mathcal{V}}(\beta \cdot x \cdot sl')\big) \in B$$

there are $x' \in X$ and $\alpha' \in Ev^*$ such that

$$\beta \cdot x' \cdot \alpha' \in [\![LTS]\!] \wedge O_{\mathcal{V}}(x' \cdot \alpha') = O_{\mathcal{V}}(\alpha) \wedge S_{\mathcal{V}}(x' \cdot \alpha') = S_{\mathcal{V}}(x \cdot sl')$$

This property is sufficient to match the neutral events of one component in traces of the other. However, there is still a problem if *both* components have neutral interface events that require mutual synchronization: assume we encounter a pair observable communication events in the two traces that we have to synchronize, such as $m!2$ and $m?0$ in Figure 3.3. In order to "close the zipper", we first have to synchronize any neutral communication events that precede the observable communication, e.g. $h!1$ in t_2. We match the neutral events of one trace by inserting corresponding events into the other, and vice versa. However, each insertion in t_1 might lead to additional neutral communication events in the resulting alternative trace, which have to be inserted into t_2 again, possibly leading to even more neutral events there, and so on, possibly trapping us in an infinite interaction loop. In order to solve this problem, we require that in this situation one of the systems supports insertion without attempting to communicate using neutral events, at least not *immediately*, i.e., before the next observable or secret event. We formalize this using the following property, which allows us to restrict the set of events that a system is allowed to use *between* the inserted event and the next observable or secret one. This restriction is specified using a parameter Y, the set of events that may *not* occur between an inserted event (drawn from the set of events X) and the next observable or secret event (drawn from the set of events Z). Intuitively, whenever the system is about to produce a Z-event, it must also accept an X-event first (at least if required by the bound) and *still* be able to produce the Z-event (or an equivalent one) directly afterwards, where the only acceptable delay between X and Z may be caused by neutral events that are *not* in the set Y.

Definition 3.9. Let X, Y, and Z be sets of events with $X \subseteq Ev_{\mathcal{V}}^{sec}$.

LTS supports *insertion of X before Z without Y in between for $\mathcal{V}$ and B*, abbreviated $IB[X, Y, Z, \mathcal{V}, B]$, iff for all $\beta, \alpha \in Ev^*$, $x \in X$, $z \in Z$, and $sl' \in (Ev_{\mathcal{V}}^{sec})^*$ such that

$$\beta \cdot z \cdot \alpha \in [\![LTS]\!] \wedge \big(S_{\mathcal{V}}(\beta \cdot z \cdot \alpha), S_{\mathcal{V}}(\beta \cdot x \cdot z \cdot sl')\big) \in B$$

there are, $x' \in X$, $z' \in Z$, $ys \in (N_{LTS}^{\mathcal{V}} \setminus Y)^*$, and $\alpha' \in Ev^*$ such that

$$\beta \cdot x' \cdot ys \cdot z' \cdot \alpha' \in [\![LTS]\!] \wedge O_{\mathcal{V}}(x' \cdot z' \cdot \alpha') = O_{\mathcal{V}}(z \cdot \alpha) \wedge S_{\mathcal{V}}(x' \cdot z' \cdot \alpha') = S_{\mathcal{V}}(x \cdot z \cdot sl')$$

If $Y = \emptyset$, we just say *LTS* supports insertion of X before Z for $\mathcal{V}$ and B for brevity.

We require this property to hold for the insertion of events in $\overline{N_j}$ directly before observable or secret communication events, with only local (non-communicating) events allowed in between. It is sufficient if, for any given pair of observable or secret communication events, *one* of the components satisfies this property. Hence, we allow splitting the set of interface events into two sets Z_1 and Z_2 for LTS_1 and LTS_2, respectively, such that the two combined cover the whole interface, defined as follows.

Definition 3.10. Let Z_1 and Z_2 be two sets of events with $Z_i \subseteq Ev_i^{obs} \cup Ev_i^{sec}$. We say *$Z_1$ and Z_2 cover the interface between $\mathcal{V}_1$ and $\mathcal{V}_2$ via $\|$* iff

- for all $e \in Ev_i^{obs} \cup Ev_i^{sec}$ and $e' \in Ev_j^{obs} \cup Ev_j^{sec}$ with $e \parallel e'$, either $e \in Z_i$ or $e' \in Z_j$ holds, and
- for all $e \approx_{\mathcal{V}_i} e'$, it holds that $e \in Z_i \longleftrightarrow e' \in Z_i$.

Putting it all together, the following conditions are sufficient to guarantee that the synchronization of neutral communication events is always possible.

Definition 3.11. Let $\mathcal{V}$ and $\mathcal{V}'$ be views on Ev and Ev', respectively. A system *LTS accepts the neutral events of $\mathcal{V}'$ via $\|$ in $\mathcal{V}$* iff there is a B such that

- B is total in $\overline{N'} = \big\{e \in Ev \mid \exists e' \in Ev'.\ (e \parallel e' \vee e' \parallel e) \wedge e' \notin Ev_{\mathcal{V}'}^{obs} \cup Ev_{\mathcal{V}'}^{sec}\big\}$ and
- *LTS* satisfies BD Security and supports eager insertion of $\overline{N'}$ w.r.t. $\mathcal{V}$ and B.

If in addition, for a set of events Z, *LTS* supports the insertion of $\overline{N'}$ before Z without *If* in between for $\mathcal{V}$ and B, then we say *LTS silently accepts the neutral events of $\mathcal{V}'$ before Z via $\|$ in $\mathcal{V}$*.

Two systems LTS_1 and LTS_2 *accept each other's neutral events in $\mathcal{V}_1$ and $\mathcal{V}_2$ via $\|$* iff

- $\mathcal{V}_1$ and $\mathcal{V}_2$ are composable w.r.t. $\|$,
- each LTS_i accepts the neutral events of $\mathcal{V}_j$ via $\|$ in $\mathcal{V}_i$, and

- if both $N_1 \cap If_1$ and $N_2 \cap If_2$ are non-empty, then there are Z_1 and Z_2 such that
  - Z_1 and Z_2 cover the interface between $\mathcal{V}_1$ and $\mathcal{V}_2$ via $\|$, and
  - each LTS_i silently accepts the neutral events of $\mathcal{V}_j$ before Z_i via $\|$ in $\mathcal{V}_i$.

Consider again the example of Figure 3.3. Let us assume that both systems allow the eager insertion of input events on the channel h (corresponding to neutral output events on h), and the second system supports this insertion before input events on the channel m without communication in between. This allows us to first eagerly insert $h?0$ into t_1 immediately after $h?0$ and before $m!2$. Any new neutral communication events appearing in t_1' before $m!2$ can then be inserted into t_2 without causing communication before $m?0$. Hence, we can "close the zipper" a bit further, up to the pair of communication events on channel m.

3.3.2. Matching of Observations and Secrets

The events $m!2$ and $m?0$ in Figure 3.3 illustrate the next challenge. Due to the different messages they transport, these events do not synchronize, but their observations and secrets do: Recall that, in the view that we use here, the contents of messages on output events on encrypted channels such as m are ignored and replaced by a dummy message in the secrets and observations. This is intentional, since we want to allow components the flexibility to choose their confidential outputs. For compositionality, this flexibility needs to be supported by the other component. In the situation of Figure 3.3, the second component, which has m as an input channel, provides this support by guaranteeing that the content of messages it receives on m do not interfere with its observations. This allows us to *replace* $m?0$ in t_2 by $m?2$, which synchronizes with $m!2$ in t_1.

Speaking more broadly, this is an example of a situation where the *granularity* of secrets and observations differs at the interface between two systems. Since we already assume that the view on input events is sufficiently fine-grained to fully capture the synchronization behavior (cf. Definition 3.7), only coarse-grained observations and secrets in *output* events remain as a challenge. For example, all output events on the encrypted channel m yield the same observation and secret, namely $m!\mathtt{d}$. In order to match such an observation $m!\mathtt{d}$, the receiving system must not discriminate on the message content, since the concrete output event emitted by the sender may contain any message: When the receiving system is ready to accept a message on m, it must also be ready to accept any other message on m. We formalize this by requiring that the receiving system supports the *replacement* of one input event by another, if they match with view-equivalent output events, e.g. the replacement of $m?0$ by $m?2$. We capture this formally in the following relation.

Definition 3.12. Two events $e_i, e_i' \in Ev_i$ *match different, $\mathcal{V}_j$-equivalent outputs of* LTS_j *via* $\|$, denoted $e_i \bowtie_{i,j} e_i'$, iff $e_i, e_i' \in In_i$ and there are $e_j, e_j' \in Out_j$ such that

- $e_j \parallel e_i$, $e_j' \parallel e_i'$, and $e_j \approx_{\mathcal{V}_j} e_j'$, but

- $e_j \nparallel e_i'$ or $e_j' \nparallel e_i$.

These conditions are satisfied for $m?0$ and $m?2$, with $m!2$ and $m!0$ as witnesses (assuming that LTS_1 and LTS_2 can produce these input and output events, respectively). Hence, $m?0 \bowtie_{1,2} m?2$ holds. We want the bounds B_i to support the replacement of events that are related by $\bowtie_{i,j}$.

Definition 3.13. Let $\mathcal{V}$ be a view, and let R be a relation on Ev^{sec}. The declassification bound B on $\mathcal{V}$ *includes* R iff, for all $t, t', t_0, t_0' \in (Ev^{sec})^*$ with $t_0 \, R \, t$ and $t_0' \, R \, t'$, it holds that

$$(S_{\mathcal{V}}(t), S_{\mathcal{V}}(t')) \in B \longleftrightarrow (S_{\mathcal{V}}(t_0), S_{\mathcal{V}}(t_0')) \in B$$

We combine this with the totality in neutral events as discussed in the previous subsection, and define the following notion of composable bounds.

Definition 3.14. Two declassification bounds B_1 and B_2 are *composable w.r.t.* $\mathcal{V}_1$, $\|$, *and* $\mathcal{V}_2$ iff, for each B_i, the following hold:

- B_i is total in $\overline{N_j}$ and
- B_i includes the reflexive closure of $\bowtie_{i,j}$.

This allows us to formulate the replacement of events according to $\bowtie_{i,j}$ as a BD Security property. We need a few assumptions on the local views in order to make this property effective.

First, we require that all events that we might need to replace for the purpose of synchronization are indeed secret, making Definition 3.14 effective for specifying requirements about these events, because they appear in the secret sequences that the declassification bounds talk about. Second, we require that for each output event that we might have to react to, some matching input event actually exists on the other side. Finally, we require that *composed* secrets do not contain *too much* information. Recall that, in order to prove well-behavedness, we need to construct a composed trace that produces a given secret. Hence, when we make any adaptations to the component traces by replacing input events related by $\bowtie_{i,j}$, this must not change the composed secret. In other words, the composition operator $*_S$ for secrets must *hide* any replacements of input events for the purpose of synchronization. We formalize these assumptions as follows.

Definition 3.15. Two views $\mathcal{V}_1$ and $\mathcal{V}_2$ have *complementary secrets and observations w.r.t.* $\|$ *and* $*_S$ iff

1. for all $e_j \bowtie_{j,i} e_j'$, it holds that $e_j \in Ev_j^{sec} \wedge e_j' \in Ev_j^{sec}$,
2. for all $e_1, e_1' \in In_1$ and $e_2 \in Ev_2^{sec} \cap Out_2$ with $e_2 \parallel e_1$ and $e_1 \bowtie_{1,2} e_1'$, it holds that $getSec_1(e_1) *_S getSec_2(e_2) = getSec_1(e_1') *_S getSec_2(e_2)$,
3. for all $e_2, e_2' \in In_2$ and $e_1 \in Ev_1^{sec} \cap Out_1$ with $e_1 \parallel e_2$ and $e_2 \bowtie_{2,1} e_2'$, it holds that $getSec_1(e_1) *_S getSec_2(e_2) = getSec_1(e_1) *_S getSec_2(e_2')$, and

4. for all $e_i, e_i' \in Out_i$ and $e_j \in In_j$ with $e_i \approx_{\mathcal{V}_i} e_i'$, it holds that $e_i \parallel e_j$ implies that there is some $e_j' \in In_j$ with $e_i' \parallel e_j'$.

Note that, together with view composability, these constraints imply that, if an input event e_i is *not* considered secret to some degree, then it must be observable, may only synchronize with observable output events, and whether a given output event synchronizes with e_i is fully determined by its observable information.

Lemma 3.2. *Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be views on LTS_1 and LTS_2, respectively, that are composable and have complementary secrets and observations w.r.t. $\parallel$ and $*_S$.*
For all $e_i \in In_i \setminus Ev_i^{sec}$ and $e_j \in Out_j$ with $e_j \parallel e_i$, it holds that

- $e_i \in Ev_i^{obs} \setminus Ev_i^{sec}$, $e_j \in Ev_j^{obs} \setminus Ev_j^{sec}$, *and*
- $e_j' \approx_{\mathcal{V}_j} e_j$ *implies* $e_j' \parallel e_i$ *for all* $e_j' \in Out_j$.

Finally, we introduce another variant of BD Security that helps us formalize the replacement of events according to $\bowtie_{i,j}$. On the one hand, regular BD Security is too weak as discussed in the previous subsection, since it does not give us sufficient control over the position of the replacement events, and it does not preserve the prefix of the traces that have already been merged. On the other hand, eager insertion is stronger than necessary: we don't need to change the timing of events by inserting them eagerly, it is sufficient to replace them *locally*, as formalized in the following definition.

Definition 3.16. Let $R \subseteq Ev_{\mathcal{V}}^{sec} \times Ev_{\mathcal{V}}^{sec}$ be a relation on secret events.
LTS supports *local replacement via R for $\mathcal{V}$ and B*, abbreviated $LR[R, \mathcal{V}, B]$, iff for all $\beta, \alpha \in Ev^*$, $(e, e') \in R$, and $sl' \in (Ev_{\mathcal{V}}^{sec})^*$ such that

$$\beta \cdot e \cdot \alpha \in [\![LTS]\!] \wedge \big(S_{\mathcal{V}}(\beta \cdot e \cdot \alpha), S_{\mathcal{V}}(\beta \cdot e' \cdot sl')\big) \in B$$

there are $e'' \in Ev_{\mathcal{V}}^{sec}$ and $\alpha' \in Ev^*$ such that

$$\beta \cdot e'' \cdot \alpha' \in [\![LTS]\!] \wedge O_{\mathcal{V}}(e'' \cdot \alpha') = O_{\mathcal{V}}(e \cdot \alpha) \wedge S_{\mathcal{V}}(e'' \cdot \alpha') = S_{\mathcal{V}}(e' \cdot sl')$$

Intuitively, whenever the system accepts a (partially) secret event, it must also accept any R-related event instead, and this change must not interfere with any observations afterwards or anything that happened before. In summary, the following conditions are sufficient to guarantee the possibility of synchronizing the observations and secrets of two systems.

Definition 3.17. *LTS_1* and *LTS_2 accept each other's secrets and observations in $\mathcal{V}_1$ and $\mathcal{V}_2$ via $\parallel$ and $*_S$* iff

1. $\mathcal{V}_1$ and $\mathcal{V}_2$ have complementary secrets and observations w.r.t. $\parallel$ and $*_S$ and
2. there are composable bounds B_1 and B_2 such that each LTS_i support local replacement via $\bowtie_{i,j}$ for $\mathcal{V}_i$ and B_i.

Assuming our example systems satisfy these conditions, we can synchronize yet another pair of events in Figure 3.3, by locally replacing $m?0$ by $m?2$ and merging it with $m!2$.

3.3.3. Scheduling of Observations and Secrets

The final challenge is that the relative ordering of observable and secret events might not correspond to the scheduling that is expected for the alternative trace we are to construct. For example, in Figure 3.3, the event $h_2?2$ comes before $h_1?1$, but well-behavedness requires us to construct (among others) a trace where $h_1?1$ comes first. This is not directly possible with the example traces due to the synchronization on $l!2$ in between.

It turns out that we can reuse the "insertion before" property that we introduced for the mutual insertion of neutral events above in order to solve this problem. Note that, depending on the bound used, this property does not actually require the insertion of *new* events; we can also use it for the (re-)insertion of *existing* events at earlier positions in the trace. Hence, "insertion of secret events before observable events" corresponds to *moving* secret events backwards in a trace. In Figure 3.3, we can use such a property to move $h_1?1$ in front of $l!2$ in t_1, so that we can merge first $h_1?1$ into a composed trace, followed by $h_2?2$ and the communication on l.

Moving secret events backwards is not always sufficient, though. Consider a situation similar to that of Figure 3.3, but with $h_1?1$ replaced by $m_1?1$. We cannot move that event relative to $l!2$, since the observations fix their ordering. In order to construct the sequence of secrets $\langle m_1?1, h_2?2\rangle$, we therefore move $h_2?2$ *forward* in t_2. For this purpose, we formulate a property that is dual to "insertion before". It is also formulated in terms of three sets of events X, Y, and Z as additional parameters. Intuitively, whenever an X-event is followed by a Z-event in the *original* trace and the bound requires the deletion (or postponement) of the X-event, then the system can produce the Z-event more or less immediately in place of the X-event. Like in the case of "insertion before", the Z-event may only be delayed by neutral events that are *not* in Y.

Definition 3.18. Let X, Y, and Z be sets of events with $X \subseteq Ev^{sec}_{\mathcal{V}}$.

LTS supports *deletion of X before Z without using Y for $\mathcal{V}$ and B*, abbreviated $DB[X, Y, Z, \mathcal{V}, B]$, iff for all $\beta, \alpha \in Ev^*$, $x \in X$, $ns \in (N^{\mathcal{V}}_{LTS})^*$, $z \in Z$, and $sl' \in (Ev^{sec}_{\mathcal{V}})^*$ such that

$$\beta \cdot x \cdot ns \cdot z \cdot \alpha \in [\![LTS]\!] \wedge \big(S_{\mathcal{V}}(\beta \cdot x \cdot z \cdot \alpha), S_{\mathcal{V}}(\beta \cdot z \cdot sl')\big) \in B$$

there are, $x' \in X$, $z' \in Z$, $ys' \in (N^{\mathcal{V}}_{LTS} \setminus Y)^*$, and $\alpha' \in Ev^*$ such that

$$\beta \cdot ys' \cdot z' \cdot \alpha' \in [\![LTS]\!] \wedge O_{\mathcal{V}}(z' \cdot \alpha') = O_{\mathcal{V}}(z \cdot \alpha) \wedge S_{\mathcal{V}}(z' \cdot \alpha') = S_{\mathcal{V}}(z \cdot sl')$$

If $Y = \emptyset$, we just say *LTS* supports deletion of X before Z for $\mathcal{V}$ and B, for brevity.

Depending on the choice of X, Y, Z, and B, this property can be used to either actually delete an event in X from a trace, or to move it relative to the next event in Z. We use it here to define the following requirement that sequences of (purely) secret events and (purely) observable events can be rescheduled arbitrarily, by moving secret events back and forth relative to observable events. Note that since we do not actually insert or delete events, we can use the identity as the bound B.

Definition 3.19. *LTS* supports *flexible scheduling of secrets and observations for* $\mathcal{V}$ iff it supports both insertion and deletion of $(Ev^{sec}_{\mathcal{V}} \setminus Ev^{obs}_{\mathcal{V}})$ before $(Ev^{obs}_{\mathcal{V}} \setminus Ev^{sec}_{\mathcal{V}})$ for $\mathcal{V}$ and Id.

For example, for the system *Meds* and the extended view that we defined in Section 3.1.2, this condition holds trivially. Recall that, in order to achieve well-behavedness, we strengthened its notion of secrets to include information about observable events, in particular their timing. Hence, the sequence of secrets fully determines the scheduling of observations, which rules out scheduling conflicts. Formally, the set $(Ev^{obs}_{\mathcal{V}} \setminus Ev^{sec}_{\mathcal{V}})$ becomes empty, so the condition of Definition 3.19 is trivially satisfied.

In general, we need a few more requirements on the views involved in order to make this definition and those of the previous subsections effective. Intuitively, we assume that the observations and secrets contain enough information to determine whether the corresponding event is an input event, output event, or an event that is *both* observable and secret. This rules out views that, for example, assign the same observation to an input and an output event. This seems to be a rather natural requirement on views. Indeed, it happens to be satisfied by all of the views discussed in this thesis. Formally, it guarantees that, for example, when we use Definition 3.19 to move a secret input event forwards in the trace after an observable input event, then the corresponding events after the move will still be two separate input events, and not turn into outputs or collapse into one event that is both observable and secret.

Definition 3.20. Let *LTS* be a labeled transition system. A view $\mathcal{V} = (Ev^{obs}, getObs, Ev^{sec}, getSec)$ is *well-defined for LTS* iff

- for all $e, e' \in Ev^{obs}$, $getObs(e) = getObs(e')$ implies $e \in In \longleftrightarrow e' \in In$, $e \in Out \longleftrightarrow e' \in Out$, and $e \in Ev^{sec} \longleftrightarrow e' \in Ev^{sec}$, and
- for all $e, e' \in Ev^{sec}$, $getSec(e) = getSec(e')$ implies $e \in In \longleftrightarrow e' \in In$, $e \in Out \longleftrightarrow e' \in Out$, and $e \in Ev^{obs} \longleftrightarrow e' \in Ev^{obs}$.

The conditions of Definitions 3.19 and 3.20 are sufficient to solve the problem of scheduling secrets and observations not only in Figure 3.3, but generally, as we will show in Section 3.4. They are not necessary conditions, though. For example, in Chapter 4 we will discuss a compositionality result (developed in joint work by the co-authors of [BPPR17]) that does not require proving flexible scheduling of secrets and observations as defined above. Instead, it focuses on scenarios where only *one* of the components locally generates secret information—an assumption called *secret-polarization* in [BPPR17]. The scheduling of secrets is then fully determined by that component. This holds even if secrets are sent to the other component synchronously; as long as the receiving component does not locally generate secrets itself, scheduling problems cannot arise. We will discuss this assumption in more detail in Chapter 4, but already define it here (together with the dual notion of observation-polarization) in order to integrate it into our compositionality result.

Definition 3.21. LTS_1 and LTS_2 are

- *secret-polarized w.r.t.* $\mathcal{V}_1$, $\|$, *and* $\mathcal{V}_2$ iff either $Ev_1^{sec} \setminus \overline{N_2} \subseteq Ev_1^{obs} \cap If_1$ holds or $Ev_2^{sec} \setminus \overline{N_1} \subseteq Ev_2^{obs} \cap If_2$, and
- *observation-polarized w.r.t.* $\mathcal{V}_1$, $\|$, *and* $\mathcal{V}_2$ iff either $Ev_1^{obs} \subseteq Ev_1^{sec} \cap If_1$ holds or $Ev_2^{obs} \subseteq Ev_2^{sec} \cap If_2$.

Note that these conditions only refer to the views and not the systems themselves. Hence, they give us a way to solve the scheduling challenge by strengthening the views and proving standard BD Security properties for the components instead of the more complex property of flexible scheduling discussed above.

3.4. Composing BD Security for LTSs

We are now ready to combine the side conditions discussed above into a general compositionality result for LTSs. We will then illustrate that result by presenting, as a first example, a strengthened version of the PSNI-like property of Section 2.2.4 that is fully compositional. Finally, we will discuss the compositionality of the side conditions themselves, which is important for moving from binary to n-ary composition.

3.4.1. A Compositionality Result for LTSs

We now consider the composition of two arbitrary, but fixed systems LTS_1 and LTS_2 using some arbitrary, but fixed synchronization relation $\|$ on the events of the two components. Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be views on LTS_1 and LTS_2, respectively, and let $\mathcal{V}$ be their composition w.r.t. $\|$ and some arbitrary, but fixed composition operators $*_O$ and $*_S$ for observations and secrets, respectively. The composition setup we assume in this section is

$$\mathcal{CS} = (O_{\mathcal{V}_1}, S_{\mathcal{V}_1}, \|, O_{\mathcal{V}_2}, S_{\mathcal{V}_2}, O_{\mathcal{V}}, S_{\mathcal{V}}, C)$$

where the observation-secret compatibility relation C is defined as follows. Relevant for compatibility are those events that are both observable and secret (to some degree). We consider a sequence of observations and a sequence of secrets of the composed system *compatible* if they could have been produced by traces with the same scheduling of events that are both observable and secret. For this purpose, we define the set of *observations* that originate from events that are also secret (and vice versa) as follows.

$$\mathsf{Obs}_i^{\mathsf{Sec}} = \{getObs_i(e) \mid e \in Ev_i^{obs} \cap Ev_i^{sec}\}$$
$$\mathsf{Sec}_i^{\mathsf{Obs}} = \{getSec_i(e) \mid e \in Ev_i^{obs} \cap Ev_i^{sec}\}$$

The schedule of observations and secrets, respectively, is given by the two functions

$$obsSchedule(o \cdot ol) = \begin{cases} 1 \cdot obsSchedule(ol) & \text{if } o = (1, o_1) \wedge o_1 \in \mathsf{Obs}_1^{\mathsf{Sec}} \\ 2 \cdot obsSchedule(ol) & \text{if } o = (2, o_2) \wedge o_2 \in \mathsf{Obs}_2^{\mathsf{Sec}} \\ 3 \cdot obsSchedule(ol) & \text{if } o = (o_1 *_O o_2) \wedge o_i \in \mathsf{Obs}_i^{\mathsf{Sec}} \\ obsSchedule(ol) & \text{otherwise} \end{cases}$$

$$secSchedule(s \cdot sl) = \begin{cases} 1 \cdot secSchedule(sl) & \text{if } s = (1, s_1) \wedge o_1 \in \mathsf{Sec}_1^{\mathsf{Obs}} \\ 2 \cdot secSchedule(sl) & \text{if } s = (2, s_2) \wedge o_2 \in \mathsf{Sec}_2^{\mathsf{Obs}} \\ 3 \cdot secSchedule(sl) & \text{if } s = (s_1 *_S s_2) \wedge s_i \in \mathsf{Sec}_i^{\mathsf{Obs}} \\ secSchedule(sl) & \text{otherwise} \end{cases}$$

This leads us to the compatibility relation $C \subseteq \mathsf{Obs}^* \times \mathsf{Sec}^*$ with

$$(ol, sl) \in C \longleftrightarrow obsSchedule(ol) = secSchedule(sl)$$

which induces the following equivalence relation on secret sequences:

$$sl \preceq_C sl' \longleftrightarrow secSchedule(sl) = secSchedule(sl')$$

We are now ready to formalize sufficient conditions for the well-behavedness of the composition of labeled transition systems using the requirements formulated in the previous sections.

Lemma 3.3. *Let LTS be the composition of LTS_1 and LTS_2 w.r.t. $\|$, let $\mathcal{V}_1$ and $\mathcal{V}_2$ be well-defined views on LTS_1 and LTS_2, respectively, and let $\mathcal{V}$ be the composition of $\mathcal{V}_1$ and $\mathcal{V}_2$ w.r.t. $\|$, $*_O$, and $*_S$, such that $\mathcal{V}$ is well-defined for LTS. LTS is a well-behaved composition of LTS_1 and LTS_2 w.r.t. $\mathcal{CS}$ if the following conditions hold:*

1. *LTS_1 and LTS_2 accept each other's neutral events in $\mathcal{V}_1$ and $\mathcal{V}_2$ via $\|$.*
2. *LTS_1 and LTS_2 accept each other's secrets and observations in $\mathcal{V}_1$ and $\mathcal{V}_2$ via $\|$ and $*_S$.*
3. *One of the following holds:*
   - *LTS_1 and LTS_2 are secret-polarized w.r.t. $\mathcal{V}_1$, $\|$ and $\mathcal{V}_2$.*
   - *LTS_1 and LTS_2 are observation-polarized w.r.t. $\mathcal{V}_1$, $\|$ and $\mathcal{V}_2$.*
   - *Both LTS_1 and LTS_2 support flexible scheduling of secrets and observations for $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively.*

As a trivial example, consider again the system *Meds* from Section 3.1. It does not have neutral events at the communication interface, so the first condition holds trivially. Moreover, the secrets and observations are the complete events themselves, so there is no information abstracted away by the views and therefore no need to

replace events in order to match observations and secrets (formally, the relations $\bowtie_{i,j}$ are empty). Hence, the second condition holds trivially, as well. Finally, the third condition is also satisfied, since we have extended the view for the components such that $Ev^{obs'}_{Med} \setminus Ev^{sec'}_{Med} = \emptyset$ holds. The well-behavedness of that system therefore follows trivially from Lemma 3.3.

Once we have managed to prove well-behavedness of the composition of two labeled transition systems, compositionality of BD Security follows immediately as an instance of Theorem 3.1.

Theorem 3.4. *Let LTS be a well-behaved composition of LTS_1 and LTS_2 w.r.t. $\mathcal{CS}$, let $\mathcal{V}_1$ and $\mathcal{V}_2$ be two views on LTS_1 and LTS_2, respectively, and let $\mathcal{V}$ be their composition w.r.t. $\|$, $*_O$, and $*_S$.*

If each LTS_i satisfies BD Security w.r.t. $\mathcal{V}_i$ and B_i, then LTS satisfies BD Security w.r.t. $\mathcal{V}$ and any $B \subseteq B_1 \|_B B_2$.

Note that this theorem does not make any restrictions on the bounds B_1 and B_2. Hence, its strength is that it allows us to compose security properties with arbitrary declassification policies. This comes with caveats, though. First, we need to prove side conditions in order to be able to use the theorem. This is not specific to BD Security, but it is something that we need to take into account when we try to build systems with BD Security guarantees in a compositional way. Second, the above theorem only talks about *two* systems. In order to move to n-ary composition, we also need to consider the preservation of the *side conditions* under composition. We will discuss this in Section 3.4.3. Third, we need to be aware that composition might lead to more confidential information being leaked than by the components themselves, due to synchronization. It is therefore prudent to check after composition that the bound B indeed captures the desired declassification policy. This is complicated by the fact that strengthening the security properties of the components in order to make them compositional (due to the first caveat) might change the shape of those properties, e.g. the notions of secrets, which makes comparisons harder. To solve this problem, we will formulate a notion of translation between security properties with different shapes of events, observations, and secrets in Section 4.3.2. We will illustrate this notion as well as the side conditions for compositionality in a number of examples throughout this thesis. We begin by strengthening the PSNI-like property of Section 2.2.4 in order to make it compositional.

3.4.2. An Example of a Compositional Security Property

Recall that, in Section 2.2.4, we considered systems communicating on channels that are partitioned into non-confidential "low" channels (L), encrypted channels (M), and secret channels (H). For the latter, we consider both the occurrence and the content of messages as confidential, while for messages on M-channels, occurrence is observable, and only the content is to be kept confidential. We formalized this as a BD Security property with a view $\mathcal{V}_{L,M,H}$ and a bound $B_{L,M,H}$ that requires the arbitrary replacement of the content of messages on M-channels and the arbitrary

insertion and deletion of messages on H-channels, without interfering with observations on L-channels. We impose these requirements only on *input* messages, since we do not want to restrict the output behavior of systems by prescribing a fixed (in)dependence of outputs and inputs—after all, this security property is supposed to be applicable to arbitrary systems.

As we have seen in Section 3.2.3, our original formulation of this property is not compositional. We have already discussed ways to address the compositionality problems in Section 3.3, and have formulated strengthened properties serving as sufficient side conditions for the compositionality of BD Security. We will shortly prove that those properties *themselves* can be composed, but this requires one more technical notion. Note that the properties we presented in Section 3.3, e.g. eager insertion, share a common structure: they are all formulated in a *backwards-strict* manner. They require the system to accept specific events at specific positions in a trace, and allow the system to adapt the trace *after* that position, but not *before*. Intuitively, this principle of backwards-strictness is useful to protect against certain active attackers, e.g. Trojan horses, that try to manipulate the system execution at runtime in order to leak confidential information to observers. Backwards-strictness guarantees that the system can always avoid information leaks by adapting only its future behavior in a strictly causal way.[4] The following property requires a system to support *all* changes of secret information specified by the declassification bound in a backwards-strict manner.

Definition 3.22. *LTS* supports *backwards-strict BD Security for $\mathcal{V}$ and B*, abbreviated *BSBD*, iff for all $\beta, \alpha \in Ev^*$ and $sl' \in \mathsf{Sec}^*$ such that

$$\beta \cdot \alpha \in [\![LTS]\!] \wedge \big(S_{\mathcal{V}}(\beta \cdot \alpha), S_{\mathcal{V}}(\beta) \cdot sl'\big) \in B$$

there is $\alpha' \in Ev^*$ such that

$$\beta \cdot \alpha' \in [\![LTS]\!] \wedge O_{\mathcal{V}}(\alpha') = O_{\mathcal{V}}(\alpha) \wedge S_{\mathcal{V}}(\alpha') = sl'$$

We will show in Section 3.4.3 how this property helps with the composition of side conditions in general. First, we use it to define a compositional variant of PSNI. Let $\mathrm{Chans}(Ev) = \{c \mid \exists m.\ c!m \in Ev \vee c?m \in Ev\}$ denote the set of channels used in the set of events Ev. Let Ev_L, Ev_M, and Ev_H denote the sets of events on channels in L, M, and H, respectively, e.g. $Ev_L = \{e \mid \exists\, l \in L, m.\ e = l!m \in Ev \vee e = l?m \in Ev\}$. Similarly, let Ev_{LI}, Ev_{MI}, and Ev_{HI} denote the set of *input* events on the channels in L, M, and H, respectively, e.g. $Ev_{LI} = \{e \mid \exists\, l \in L, m.\ e = l?m \in Ev\}$.

Definition 3.23. Let $L, M, H \subseteq \mathrm{Chans}(Ev)$ be disjoint sets of channels. The system *LTS satisfies PSNI for L, M, and H* iff it satisfies BD Security and the following properties for $\mathcal{V}_{L,M,H}$ and $B_{L,M,H}$ (cf. Section 2.2.4):

- backwards-strict BD Security,

[4] Cf. [Man04, Section 3.4.4] for an example of an information leak that is only detected with backwards-strict security properties.

- eager insertion of Ev_{HI},
- insertion of Ev_{HI} before $(Ev_{LI} \cup Ev_{MI})$ without $(In \cup Out)$ in between, and
- local replacement via $\approx^{obs}_{MI}$, where $e \approx^{obs}_{MI} e'$ iff $e, e' \in Ev_{MI}$ and $e \approx^{obs}_{\mathcal{V}_{L,M,H}} e'$.

Intuitively, this property requires that the system is ready to receive high inputs at any time, and neither the contents of confidential messages nor the (non)occurrence of secret messages interfere with observations. Furthermore, the reactions of the system to high inputs are restricted by the third condition above: when the system is ready to accept observable input, it must still be ready to accept it immediately after accepting a sequence of high inputs, i.e., without insisting on any other communication in between. In particular, it must not insist on sending an output, which could block the observable communication event, if the other system is not ready to receive at the same time.

These additional side conditions allow us to instantiate the compositionality result of Theorem 3.4 as follows. Assume we want to compose two systems LTS_1 and LTS_2 and their PSNI properties w.r.t. L_i, M_i, and H_i for $i \in \{1, 2\}$. We assume that the systems communicate on some shared channels, and that they agree on the classifications of those channels: a shared channel is in L_1 iff it is in L_2, and similarly for M_i and H_i. Moreover, we assume that the systems agree on the possible contents of messages that may be exchanged, i.e., if c is an output channel of LTS_1 and an input channel of LTS_2, then for every message m, $c!m \in Out_1$ iff $c?m \in In_2$.

We now apply Lemma 3.3 to derive the well-behavedness of the composition of LTS_1 and LTS_2. Its first condition is that the systems accept each other's neutral events in the sense of Definition 3.11. It is easy to check that this condition is satisfied, assuming that the additional side conditions we impose for PSNI hold. In particular, note that the sets $\overline{N_i}$ contain exactly the high input events at the interface between LTS_1 and LTS_2, since only the corresponding high outputs are neutral. Each LTS_i eagerly accepts those events, and it silently accepts them before observable inputs, which cover the interface between LTS_1 and LTS_2. Hence, the systems LTS_i satisfy the conditions of Definition 3.11 regarding the insertion of neutral events. The conditions on bounds and views are satisfied, too, at least if we additionally assume that the interface between LTS_1 and LTS_2 only consists of events on the channels L, M, and H. In this case, all input events at the interface are observable or secret, and the $getSec$ function on inputs is the identity, so the conditions in Definition 3.7 are satisfied. Moreover, the bounds B_{L_i,M_i,H_i} are total in all secret events.

The second condition for well-behavedness is that the systems accept each other's secrets and observations. In addition to neutral events, there is another asymmetry between the views: the contents of encrypted output messages are stripped from secrets and observations, while for the corresponding input messages, the content is preserved in the secrets (but not in the observations). This means that, in case of a mismatch in the content of output and input events, we might have to replace the input event by one that matches the output. Formally, Definition 3.17 requires us to

prove that we can locally replace events according to the relation $\bowtie_{i,j}$, which in this case relates an input event on an encrypted channel with all other input events on the same channel. Hence, this condition is satisfied by the fourth property above, together with the fact that the bounds B_{L_i,M_i,H_i} indeed include these replacements of encrypted inputs. In order to show that the views $\mathcal{V}_{L_i,M_i,H_i}$ have complementary secrets and observations in the sense of Definition 3.15, we choose a secret combination operator $*_S$ that hides the variance of inputs related by $\bowtie_{i,j}$ (i.e., input events on channels in M):

$$getSec_1(e_1) *_S getSec_2(e_2) = \begin{cases} getSec_1(e_1) & \text{if } e_1 \parallel e_2 \\ getSec_2(e_2) & \text{otherwise} \end{cases}$$

This operator always chooses the secret produced by the output event, which is $c!\mathtt{d}$ for outputs on channels in M. It only contains a dummy value and is independent of the concrete input; it is then easy to check that the conditions of Definition 3.15 are satisfied.

Finally, the third condition of Lemma 3.3 regarding the flexible scheduling of secrets and observations is implied by the side conditions of Definition 3.23. We can delete any high inputs that are in the wrong position using backwards-strict BD Security in combination with totality of the bound, and then re-insert them at the position where we need them using eager insertion of high inputs.

Hence, we can apply Theorem 3.4 to derive a security property of LTS from the security of LTS_i. However, there is a technical complication: the resulting security property has a different shape than the original PSNI properties, because LTS uses composite events such as $(1, l!m)$ or $(h!m, h?m)$ due to the definition of our composition operator. To solve this problem, consider the following translation of events:

$$r(e) = \begin{cases} e_1 & \text{if } e = (1, e_1) \vee (e = (e_1, e_2) \wedge e_1 \in Out_1) \\ e_2 & \text{if } e = (2, e_2) \vee (e = (e_1, e_2) \wedge e_1 \notin Out_1) \end{cases}$$

If we assume, for simplicity, that $Ev_1 \cap Ev_2 = \emptyset$ holds, i.e., the event sets of LTS_1 and LTS_2 are disjoint, then r is a bijection between the sets Ev and $(Ev_1 \cup Ev_2) \setminus ((If_1 \cap In_1) \cup (If_2 \cap In_2))$: Local events are recovered in their original form, while synchronization events are represented by the participating *output* event. Note that, for the composition setup we use for PSNI, the output event in $(e_1, e_2) \in Ev$ uniquely determines the input event: either $e_1 = c!m$, then $e_2 = c?m$, or vice versa. Intuitively, this simplifying assumption $Ev_1 \cap Ev_2 = \emptyset$ is satisfied, in particular, if the channels used by both systems are unidirectional: they are used by one system for input, and by the other system for output. Given that bidirectional channels can be split into two unidirectional channels with distinct names, this assumption is without loss of generality. BD Security is preserved under the bijective translation r. (We will present a general translation theorem for BD Security in Section 4.3.2.) This allows us to apply the general compositionality result first, and then translate both system and security property back to the familiar shape of PSNI.

Theorem 3.5. *Let LTS_1 and LTS_2 be two systems where $Ev_1 \cap Ev_2 = \emptyset$, $\mathrm{Chans}(In_i) \cap \mathrm{Chans}(Out_i) = \emptyset$, and for all messages m on shared channels $c \in \mathrm{Chans}(Ev_1) \cap \mathrm{Chans}(Ev_2)$, it holds that $c!m \in Out_i$ iff $c?m \in In_j$. Let LTS be the composition of LTS_1 and LTS_2 w.r.t. $\|$. Let each LTS_i satisfy PSNI w.r.t. L_i, M_i, and H_i such that*

$$\begin{aligned} L_1 \cap \mathrm{Chans}(If_1) &= L_2 \cap \mathrm{Chans}(If_2) \\ M_1 \cap \mathrm{Chans}(If_1) &= M_2 \cap \mathrm{Chans}(If_2) \\ \mathrm{Chans}(In_i) &\subseteq L_i \cup M_i \cup H_i \end{aligned}$$

Then $r(LTS)$ satisfies PSNI w.r.t.

$$\begin{aligned} L &= L_1 \cup L_2 \\ M &= M_1 \cup M_2 \\ H &= (H_1 \setminus \mathrm{Chans}(If_2)) \cup (H_2 \setminus \mathrm{Chans}(If_1)) \end{aligned}$$

Intuitively, the composed system still satisfies PSNI w.r.t. secret inputs that are received from the *environment*. We assume that the classifications of channels of the two components agree: interface channels that are observable or encrypted for one system are observable or encrypted, respectively, for the other system. Remaining interface channels of one system are assumed to be high input channels of the other. These high channels that are used internally for the synchronization between the components become neutral channels of the composed system.

3.4.3. Preservation of Side Conditions

There is one more ingredient we need in order to finish the proof of PSNI's compositionality: We need to show that the composed system again satisfies the side conditions that we imposed for PSNI, i.e., the eager insertion of confidential inputs and the insertion before observable events without outputs in between.

All of these properties share a similar structure. Like plain BD Security, they require that it is possible to change the secret without interfering with the observations. However, they are more specific in *what* changes are required *where* in a trace, e.g. the insertion of a secret event directly before an observable event, and *how* the result after the change may look like, e.g. that the secret event must be inserted at exactly the same position in which the observable event was before. For all these properties, we can formally capture the specific requirements in terms of two predicates P and Q, formalizing the details of what has to hold before and after changing the secret, respectively. The position of the change in the trace is captured by splitting the trace into a part β before the change and a part α after the change. The properties have in common that β is required to stay fixed, i.e., the system is only allowed to react causally *after* the change, without having to adapt the past, i.e., in a backwards-strict manner. In addition to β and α, we parameterize the predicate P by the sequence of secrets that has to be produced after β, and the

Table 3.1.: Side conditions

Condition[a,b]	$P(\beta, \alpha, sl')$ [c]	$Q(\beta, \alpha, \alpha')$ [c]
BSBD	True	True
$EI[X]$	$getSec(x) \leq sl'$	$x' \leq \alpha'$
$LR[R]$	$s\, R\, s' \wedge s \leq \alpha \wedge getSec(s') \leq sl'$	$s' \leq \alpha'$
$IB[X, Y, Z]$	$z \leq \alpha \wedge S_{\mathcal{V}}(\langle x, z \rangle) \leq sl'$	$x' \cdot ys' \cdot z' \leq \alpha' \wedge z' \approx_{\mathcal{V}} \mathrm{hd}(\alpha)$
$DB[X, Y, Z]$	$x \cdot ns \cdot z \leq \alpha \wedge S_{\mathcal{V}}(\langle z \rangle) \leq sl'$	$ys' \cdot z' \leq \alpha' \wedge z' \approx_{\mathcal{V}} \mathrm{hd}(\alpha \restriction Z)$

[a] where we omit the parameters $[\ldots, \mathcal{V}, B]$ for brevity
[b] where $\mathcal{V} = (Ev^{obs}, getObs, Ev^{sec}, getSec)$, $X \subseteq Ev^{sec}$, $Y \subseteq N^{\mathcal{V}}_{LTS}$, and $Z \subseteq Ev^{obs} \cup Ev^{sec}$
[c] Free variables in each cell are existentially quantified, such that s, s' range over Ev^{sec}, ns over $(N^{\mathcal{V}}_{LTS})^*$, x, x', x'' over $X \cap Ev^{sec}$, z, z' over $Z \cap (Ev^{obs} \cup Ev^{sec})$, and ys, ys' over $(N^{\mathcal{V}}_{LTS} \setminus Y)^*$.

predicate Q by the alternative trace α' produced by the system, in order to check the additional constraints before and after the trace.

Definition 3.24. Let P and Q be two predicates of the types

$$P\colon Ev^* \times Ev^* \times (Ev^{sec}_{\mathcal{V}})^* \to Bool$$
$$Q\colon Ev^* \times Ev^* \times Ev^* \to Bool$$

We say the system *LTS* satisfies the BD security property with the side condition (P, Q) w.r.t. a view $\mathcal{V}$ and a bound B iff for all $\beta, \alpha \in Ev^*$ and $sl' \in (Ev^{sec}_{\mathcal{V}})^*$ such that

$$\beta \cdot \alpha \in [\![LTS]\!] \wedge \left(S_{\mathcal{V}}(\beta) \cdot S_{\mathcal{V}}(\alpha), S_{\mathcal{V}}(\beta) \cdot sl'\right) \in B \wedge P(\beta, \alpha, sl')$$

there is an $\alpha' \in Ev^*$ such that

$$\beta \cdot \alpha' \in [\![LTS]\!] \wedge O_{\mathcal{V}}(\alpha') = O_{\mathcal{V}}(\alpha) \wedge S_{\mathcal{V}}(\alpha') = sl' \wedge Q(\beta, \alpha, \alpha')$$

Table 3.1 summarizes the properties we presented in Section 3.3, formulated in terms of side conditions. They can be locally verified for a system component using the unwinding technique discussed in Appendix B. In this section, we show that these properties are also preserved under composition, if certain conditions hold. In particular, we require that *BSBD* holds for the components in addition to the property we want to compose. Even though all the properties we have presented include some form of backwards-strictness, this is not sufficient for compositionality. For example, consider the compositionality of *LR*. Assume we have to prove that it is possible to replace an event in the trace of the composed system that is originally a local event of one of the components. We apply the *LR* property of that component to perform the local replacement. However, the declassification bound might require additional changes after the replacement, including changes in the trace of the *other* component. In order to perform those changes in the other trace, we apply the *BSBD* property of that component. Finally, we "close the zipper" again by applying well-behavedness, synchronizing the unmatched suffixes of the component traces.

Table 3.2.: Composition of side conditions

Component properties[a]	Constraints	Result
BSBD	–	*BSBD*
$EI[X_i]$, *BSBD*	$X_i \cap Ev_i^{obs} = \emptyset$	$EI[X]$
$LR[R_i]$, *BSBD*	–	$LR[R]$
$IB[X_i, Y_i, Z_i]$, $EI[X_i]$, *BSBD*	$X_i \cap Ev_i^{obs} = \emptyset$, $If_i \subseteq Y_i$	$IB[X, Y, Z]$
$DB[X_i, Y_i, Z_i]$, *BSBD*	$X_i \cap Ev_i^{obs} = \emptyset$, $\overline{N_j} \subseteq X_i$, $If_i \subseteq Y_i$	$DB[X, Y, Z]$

[a] where we omit $[\ldots, \mathcal{V}_i, B_i]$ from the parameters for brevity

Table 3.2 summarizes the combinations of security properties that are composable. For example, assume we want to show that the composed system satisfies $IB[X, Y, Z, \mathcal{V}, B]$ from corresponding component properties $IB[X_i, Y_i, Z_i, \mathcal{V}_i, B_i]$, where X_i, Y_i, and Z_i are the projections of X, Y, and Z to the events of respective component. According to Table 3.2, we additionally require the components to satisfy backwards-strict BD Security, as well as the eager insertion of X_i. This allows us, in particular, to insert a local event $x \in X_1$ before a local event $z \in Z_2$ into the trace of the composed system: We apply the *EI* property of the first component to insert x, and the *BSBD* property of the second component to perform any remaining changes in the trace after z.

Table 3.2 also formulates constraints on the parameters X_i, Y_i, and Z_i for the insertion and deletion properties. First, we assume that the sets X_i do not contain observable events. We do not consider this to be a serious restriction, since it generally makes little sense to demand the insertion of observable events. We rule it out explicitly here, since attempting to insert a local observable event of one component ahead of schedule can interfere with the scheduling of observable events in the composition, causing the proof of compositionality to fail. Note that we can still formulate requirements about the scheduling of observable events, via the insertion or deletion of the *secret* events around them (as we have done in Definition 3.19, for example). In the case of *DB*, we make another assumption regarding X_i, namely that the properties are strong enough to support the deletion of events in $\overline{N_j}$, i.e., communication events corresponding to neutral events of the other component. This guarantees that, for the composed system, the possibility of deleting (other) X-events before Z is not blocked by the systems insisting on performing neutral communication in between.

In a similar vein, we assume $If_i \subseteq Y_i$, i.e., we only consider the composition of *IB* and *DB* properties that do not allow communication events immediately after an inserted or deleted X-event. This guarantees that the possibility of producing the next Z-events is not blocked by the presence of neutral communication between the components.

It is easy to check that the above constraints hold for the side conditions of PSNI in Definition 3.23. Hence, we can indeed compose them. The final question is what

$$
\begin{aligned}
B_1 \parallel_B^{BS} B_2 = \{(sl, sl') \mid \forall \beta, \alpha, \alpha', \beta_1, \beta_2, \alpha_1, \alpha_2.\ & \\
& sl = \beta \cdot \alpha \wedge sl' = \beta \cdot \alpha' \wedge \\
& \beta \in \beta_1 \parallel_S \beta_2 \wedge \alpha \in \alpha_1 \parallel_S \alpha_2 \\
\longrightarrow (\exists \alpha_1', \alpha_2'.\ & \alpha' \in \alpha_1' \parallel_S \alpha_2' \wedge \\
& (\beta_1 \cdot \alpha_1, \beta_1 \cdot \alpha_1') \in B_1 \wedge \\
& (\beta_2 \cdot \alpha_2, \beta_2 \cdot \alpha_2') \in B_2 \wedge \\
& (\beta \cdot \alpha) \preceq_C (\beta \cdot \alpha'))\}
\end{aligned}
$$

Figure 3.4.: Backwards-strict composition of bounds

the exact security property is that we obtain from this composition, in particular the declassification bound. It turns out that, in general, the composition operator $\parallel_B$ for declassification bounds is not specific enough for composing the properties of Table 3.1: it ignores the fact that all those properties are *backwards-strict*. Hence, in order to leverage compositionality, the composed declassification bound needs to guarantee that we can decompose pairs of secret sequences in a *backwards-strict* manner so that we can invoke the corresponding properties of the components as needed. For this purpose, we modify Definition 3.4 so that common prefixes of secret sequences are preserved by the decomposition.

Definition 3.25. The *backwards-strict composition* of B_1 and B_2 w.r.t. $\mathcal{CS}$ is defined as given in Figure 3.4.

This is a refinement of the original composition operator for bounds, i.e. $B_1 \parallel_B^{BS} B_2 \subseteq B_1 \parallel_B B_2$. The converse direction does not hold in general. One particular case where it holds are bounds that are history-independent.

Definition 3.26. A declassification bound B on $\mathcal{V}$ is *history-independent* iff, for all $\beta, \alpha, \alpha' \in \mathsf{Sec}^*$, it holds that $(\beta \cdot \alpha, \beta \cdot \alpha') \in B \longleftrightarrow (\alpha, \alpha') \in B$.

Lemma 3.6. *Let $B \subseteq B_1 \parallel_B B_2$. If B, B_1, and B_2 are history-independent, then $B \subseteq B_1 \parallel_B^{BS} B_2$.*

With a backwards-strict composition of bounds, the properties of Table 3.1 are preserved under composition in certain combinations, summarized in Table 3.2. The parameters X, Y, Z, and R are composed by essentially taking the union of the component parameters. We make one additional assumption on the sets X_i and Z_i, namely that they are defined in a way that does not depend on details of communication events that are abstracted away by the composed view. This guarantees that the synchronization of X- or Z-events (possibly changing communication details that are not reflected in observations and secrets) does not invalidate the property.

Theorem 3.7. *Let $B \subseteq B_1 \parallel_B^{BS} B_2$, let $X_i, Y_i, Z_i \subseteq Ev_i$ and $R_i \in Ev_i \times Ev_i$, and let*

$$
\begin{aligned}
X &= (\{1\} \times X_1) \cup (\{2\} \times X_2) \cup (X_1 \times X_2) \\
Y &= (\{1\} \times Y_1) \cup (\{2\} \times Y_2) \\
Z &= (\{1\} \times Z_1) \cup (\{2\} \times Z_2) \cup (Z_1 \times Z_2) \\
R &= \big\{\big((1,e_1),(1,e_1')\big) \mid (e_1,e_1') \in R_1\big\} \cup \\
&\quad \big\{\big((2,e_2),(2,e_2')\big) \mid (e_2,e_2') \in R_2\big\} \cup \\
&\quad \big\{\big((e_1,e_2),(e_1',e_2')\big) \mid (e_1,e_1') \in R_1 \wedge (e_2,e_2') \in R_2 \wedge \\
&\qquad \big((e_1 \parallel e_2 \wedge e_1' \parallel e_2') \vee (e_2 \parallel e_1 \wedge e_2' \parallel e_1')\big)\big\}
\end{aligned}
$$

If

- *the preconditions of Lemma 3.3 hold,*
- *the bounds B_1 and B_2 are composable w.r.t. $\mathcal{V}_1$, $\parallel$, and $\mathcal{V}_2$ (cf. Definition 3.14),*
- *the constraints in the middle column of Table 3.2 hold,*
- *$e \approx_{\mathcal{V}} e'$ implies $e \in X \longleftrightarrow e' \in X$ and $e \in Z \longleftrightarrow e' \in Z$, and*
- *each LTS_i satisfies the security properties in the left column of Table 3.2 w.r.t. $\mathcal{V}_i$ and B_i,*

then LTS satisfies the security property in the right column of that row of the table w.r.t. $\mathcal{V}$ and B.

For detailed proofs for this and the other theorems, see Appendix C. This theorem allows us to finish the proof of compositionality for PSNI. In the following chapters, we will discuss instantiations of compositionality of BD Security with more complex security requirements than PSNI.

3.5. Related Work

Research in information flow security has a rich history, mostly unfolded from the pioneering notions of noninterference [GM84] and nondeducibility [Sut86] (of which BD Security is a generalization). For discussing the declassification aspects of related work, we employ the well-known taxonomy due to Sabelfeld and Sands [SS09], which is particularly relevant for the rich information exchange in realistic applications: Ideally, security policies should be able to describe precisely *what* information can be released (e.g., the content of a document), *by whom* (e.g., the owners of the document), as well as *where* and/or *when* (e.g., only after the document is marked by the owner as public). We focus on the compositionality of the various approaches.

Language-based approaches [SM03] concentrate on specific programming languages. Since they operate on the level of language syntax, they often achieve an

impressive degree of automation. For example, Jif [Mye99] extends Java with security labels for data values and enforces security via a combination of static and run-time checking. It supports control over *who* may declassify information, but not *what* is declassified. Joana [HS09] checks noninterference of Java programs via static program analysis. Control of declassification is limited to *where* in the program declassification may occur. Jeeves [YYS12] extends a core language for functionality with a language for flexible security policies. RF* [BFG$^+$14] uses a relational Hoare logic to reason about 2-safety properties of probabilistic programs, including language-based notions of information flow security. The What&Where security property [LMP12] allows control over *what* is declassified by concurrent programs, but only in a non-interactive setting, not suitable for the interactive multi-user systems we will consider in the next chapters. Secure multi-execution [DP10, RS16] is a run-time enforcement technique where multiple copies of a program are executed, one for each security level, controlling the information flows between the levels. Secure program partitioning [ZZNM02, CLM$^+$09] produces a distribution of the code and data in a program onto different hosts according to their different trust levels (e.g., trusted web server and untrusted client-side browser).

While most of the aforementioned approaches support certain forms of declassification, they do not consider the setting and the kind of compositional reasoning that we have aimed for, i.e. deriving a global, complex information flow security property from local properties of a given set of communicating systems. Instead, the preservation of security under composition (or, similarly challenging, *refinement* [Mor09]) is often considered w.r.t. language constructs, such as sequential composition, if-then-else branches, or while loops, for example in [MR07] or [PHN13], where different requirements and assumptions w.r.t. termination and concurrency are shown to lead to different syntactic criteria for security.

DStar [ZBM08] and Fabric [LGV$^+$09] do consider distributed systems, but they offer only coarse-grained control over what information may be declassified: an assignment of security labels to both data items and principals specifies which principals may declassify which data item. In contrast, the declassification bounds of BD Security specify what aspects of confidential information may be declassified on a semantic level, formulated in terms of arbitrary logical formulas.

Our framework falls into the category of *system-based* approaches. They work with security properties expressed directly on the semantics, on variants of event systems or I/O automata. Early work in this category [McC87] has observed that even seemingly strong security properties are not preserved under composition in general. Subsequently, comprehensive frameworks have been developed for the composition of security properties in various settings, e.g., event systems [Man02], reactive systems [RS14] and process calculi [FG01, BMPR05]. Some of these focus on formulating classes of security properties that are always guaranteed to be preserved under a given notion of composition, such as McCullough's Restrictiveness [McC90], or Non-Deducibility on Composition (NDC) and its generalizations [FG01, BMPR03, MM11], which are based on the idea that a process is considered secure if it is observationally equivalent to itself composed with arbitrary high pro-

cesses. The equivalence is chosen to be strong enough so that security is preserved when composing two secure processes. Other approaches, such as Mantel's MAKS framework [Man00a, Man02], formulate side conditions on the local security properties that guarantee compositionality. The MAKS framework can capture earlier security notions such as those of McLean [McL96] and Zakinthinos and Lee [ZL97], as shown in [Man05]. Our compositionality result is inspired by that of the MAKS framework [Man02]. The composition operator of MAKS can be seen as an instance of our operator, with the difference that there is no syntactic distinction between input and output events, and systems synchronize when they perform identical, shared events; i.e. $\|$ is the identity on shared events. We have formulated strengthened notions of BD Security that are inspired by the Basic Security Predicates (BSPs) of MAKS, and we have lifted Mantel's side conditions for compositionality [Man02, Lemma 1] to BD Security; they correspond to the first condition of Lemma 3.3. The remaining two conditions of our lemma deal with some of the additional flexibility provided by BD Security, i.e., abstraction of events into observations and secrets, and scheduling flexibility. Moreover, the notion of bound composition given in Definition 3.4 allows us to compose arbitrary policies and generalizes the specific combinations of BSPs in [Man02, Theorem 3].

The above security notions allow *no* flow of confidential information to the observer (except for very specific notions such as intransitive noninterference [Man01a]), but as we discussed in Chapter 2, this is too strict already for our simple workflow example. Our examples have been mostly about the question "what" is declassified, e.g. the equivalence class of the final medical reports. In principle, the other dimensions of declassification in Sabelfeld and Sands' taxonomy can also be encoded into the declassification bound to some degree. For simple declassification policies, we could model a declassification *trigger* to fire at a certain point in time ("when") or upon an action by a certain user or component ("who" or "where"). We discuss how such triggers can be incorporated into a bound in Appendix D. More elaborate policies, e.g. releasing different aspects of confidential information at different times, could be incorporated into the bound in a more application-specific way. In the context of our case study, we discuss examples of how such dynamic declassification triggers can be encoded into the bound in Section 4.4.3.

Other approaches that consider declassification include Chong and van der Meyden's framework [CM15], where *filter functions* are used to restrict what information may flow between domains, together with an interpretation of the resulting security properties in terms of an epistemic logic. However, they do not consider compositional reasoning in our sense, i.e., composing the security properties of multiple components to a security property of the overall system. The same applies to the work on temporal logics and model checking approaches for hyperproperties [CFK$^+$14], of which information flow security properties are an instance. Greiner and Grahl [GG16] present a compositionality result that supports *what*-declassification control, specified in terms of equivalence relations on communication events, but it cannot express *dynamically changing* confidentiality requirements—as needed for many realistic systems. For example, in the discussion of a case study in Chapter 4, we

consider the confidentiality of a post in a social media platform as a running example. Whether a given post p by user u is confidential for an observer depends on the (dynamically changing) visibility setting of p and/or the friendship status between u and the observer.

The security notion from the literature that most closely resembles ours is a recent one by Guttman and Rowe [GR15], formulated on top of *blur operators*, which are similar to the declassification bounds used for BD Security. A compositionality result is presented that focuses on a different question than ours: Instead of composing the security guarantees of the individual nodes in a network, it considers a *partitioning* of the network into a core of nodes that may handle secret information and an outer part that is purely observing. In contrast, our framework deals with the question of how the security properties of the nodes *inside* the core compose to an overall confidentiality guarantee.

3.6. Conclusion

We have presented a framework for composing BD security properties, both in an abstract version and in an instantiation for labeled transition systems. In summary, our compositionality framework has the following ingredients:

- A notion of composition for declassification bounds, that allows us to merge the security properties of the subsystems to one of the composed system.
- In the case of labeled transition systems, specialized variants of BD Security that allow us to formulate side conditions for compositionality, inspired by those of the MAKS framework.
- A compositionality theorem establishing that the composition of two secure systems is indeed secure w.r.t. the composition of the declassification bounds, if the side conditions hold.

We believe that our framework combines the strengths of several existing approaches: the flexibility in specifying declassification policies of BD security [KLP14], the support for different notions of composition in the spirit of Rafnsson and Sabelfeld's library of system combinators [RS14], and compositionality results modeled after those of the MAKS framework [Man02].

Of course, this flexibility comes at a price. The side conditions of the compositionality results require us, in general, to prove stricter security properties than might seem necessary at first. The inherent complexity of compositional reasoning has led Lamport to argue that it is often more of "a way to make proofs harder" [Lam97]. However, Lamport concedes that there are cases where compositionality is very useful. It can even be necessary, in particular when *open* systems are to be verified, i.e. components that are to be used in several different contexts. For example, a PSNI component can be plugged into arbitrary, larger systems. If they satisfy PSNI, too, the whole system will satisfy a PSNI property. The case study that we present in

Chapter 4 as well as the modular development approach we discuss in Chapter 5 both deal with systems composed of an arbitrary number of components. In Chapter 6, we discuss execution monitors for safety properties that are meant to be composable with target systems that may not be known in advance. Compositionality results like the one we presented in this chapter, or the more specialized one of [BPPR17] that we discuss in Chapter 4, form the theoretical foundation for these verification efforts.

Chapter 4.

Case Study: Security of a Distributed System

This chapter describes a case study in compositional verification conducted in collaboration with Andrei Popescu, Armando Pesenti Gritti, and Franco Raimondi [BPPR17].[1] We consider a social media platform where users can submit so-called "posts" containing text and optional images. For each of these posts, they can decide whether it shall be visible publicly or only for their friends. Friendship lists can be maintained by issuing, accepting, or denying friendship requests, or (unilaterally) deleting friendship links. We have verified that

- the content of any given post is kept confidential from observing users, unless the post is currently marked as public or one of the observers is a friend of the creator of the post, and
- the information whether any two given users are friends is kept confidential unless one of the observers is currently a friend of one of them.

These properties are interesting instances of BD Security, because whether a post or friendship is confidential changes dynamically as the visibility status of the post and the friendship relations between the users change over time.

In previous work these properties had been verified for a monolithic system running on a single web server [BPPR16]. The focus of the case study discussed in this chapter is on lifting the security guarantees of a single instance of the system to a *network* of systems communicating with each other. Each of these network nodes maintains their own sets of users and content, but users can establish friendship links with remote users on other nodes, and they can read remote posts if those are public or they are a friend of the creator of the post. This is similar to existing, federated social networks such as Diaspora[2] which exists as a network of servers, where a user registers with one of the servers and can then communicate with both local and remote users.

For verifying the distributed system, we want to reuse the results of the verification of the single-system case with as little amount of changes as possible. For this

[1] This chapter is based on that paper (Copyright © 2017, IEEE). It has been edited to tie into this thesis, in particular to relate its compositionality result to that of Chapter 3.

[2] `https://diasporafoundation.org/`

purpose, we use a compositionality framework that leverages a set of assumptions that significantly simplify verification (Section 4.3). This framework was developed in joint work with the co-authors of [BPPR17] separately from the one presented in Chapter 3, but the two are closely related in that the latter incorporates and generalizes the result discussed in this chapter. In particular, the notion of secret polarization (cf. Section 3.3.3) is one of the assumptions developed in [BPPR17] and incorporated into the result of Chapter 3 as one way to solve the challenge of scheduling of observations and secrets, while the other assumptions developed in [BPPR17] imply the remaining well-behavedness conditions of Lemma 3.3. Hence, the compositionality result of [BPPR17] discussed in this chapter is not as general as Theorem 3.4, but if the simplifying assumptions hold in a given scenario, it is much easier to apply. While it might require extending views and bounds in order to satisfy the assumptions, it does *not* require proving any of the more complex side conditions of Chapter 3 such as backwards strictness or eager insertion.

We discuss this compositionality result in more detail in Section 4.3. We begin, however, by describing the concrete system and its security requirements.

4.1. System Description

This section first describes the single-instance system, called CoSMed [BPPR18], and then presents its extension to the distributed system, called CoSMeDis.

4.1.1. The Original System

The case study is about a social media platform loosely inspired by online social networks, such as Facebook. It allows users to register and post information and to restrict access to this information based on friendship relationships established between users. For example, the user Alice can log into the web site and browse through posts made by other users. She can create new posts herself, e.g., a comment on a sports event. By default, this post is visible to her friends only. She can add new friends by looking up their profile, e.g., the profile of Bob, and requesting friendship by optionally entering a greeting text and clicking the submit button. When Bob approves the request, they become friends, and Bob can now see Alice's sports comment. Alice can also edit her posts and set their visibility level—either friends-only or public—at any time. The system has one user with special privileges, called the administrator (or admin, for short), who is responsible for approving the creation of users.

System Model

CoSMed has a *state* which stores information on users, posts and friendships. For example: A user ID has an associated name, email address and info; a post ID has an associated title, text, image, owner ID and visibility; a friendship request is identified by two user IDs (the sender and the recipient) and has an associated

greeting text created by the sender; the friendship status ("friend" or "not friend") is stored as a symmetric association between user IDs.

Users interact with the system via *actions* for creating, deleting, updating and reading items in the system, where an item can be a user, a post, a friendship request or a friendship status. There are also actions for listing items according to various filters or criteria—e.g., a user can list all posts, or all posts of a given friend, or all his friends, or all the friends of a given friend, etc. Each action a also contains the user ID of its issuer, denoted by $\mathsf{userOf}(a)$.

When a user requests an action, the system checks if the action is allowed, in which case the action is applied and the output is returned to the user; otherwise, an error message is emitted and the state remains unchanged.

Formalization and Implementation

The above behavior is formalized as an automaton. It accepts *actions* from users. For example, an action that updates the content of a post has the form $(\mathsf{updatePost}, \mathit{uid}, \mathit{pid}, \mathit{pst})$. Here, $\mathsf{updatePost}$ is a label indicating the particular type of action, uid is the ID of the acting user, pid is the ID of the post, and pst is the new content. (The action parameters also include user passwords, but we omit them to simplify the notation.) Upon each action, the automaton produces an *output* (possibly an error message) and updates the state (if no error occurs).

This behavior is captured in the (deterministic) step function of the automaton, $\mathit{step}\colon \mathit{St} \times \mathit{Act} \to \mathit{Out} \times \mathit{St}$. A system *transition* is a tuple $e = (\sigma, a, o, \sigma')$. It is *valid* iff $\mathit{step}(\sigma, a) = (o, \sigma')$, i.e., the automaton produces the output o and moves to the successor state σ' when it is in the state σ and receives the action a. The states σ and σ' are called the transition's *source* and *target* states, denoted as $\mathit{src}(e)$ and $\mathit{tgt}(e)$. The transition's action and output are denoted by $\mathit{act}(e)$ and $\mathit{out}(e)$, respectively.

The automaton induces an $\mathit{LTS} = (\mathit{St}, \sigma^0, \mathit{Ev}, \emptyset, \emptyset, \to)$ whose events are the transitions, i.e.,

$$\mathit{Ev} = \mathit{St} \times \mathit{Act} \times \mathit{Out} \times \mathit{St}$$

We choose the sets of input and output events to be empty: While CoSMed communicates with its users, it does not communicate with other systems. This will change when we extend it to a distributed system below. CoSMed's transition relation is defined by

$$\sigma \xrightarrow{e} \sigma' \equiv \mathit{src}(e) = \sigma \wedge \mathit{tgt}(e) = \sigma' \wedge \mathit{step}(\sigma, \mathit{act}(e)) = (\mathit{out}(e), \sigma')$$

It might seem redundant to include the source and target states in the events. The advantage is that it allows us to define security views where the fact whether an event is secret or not is *history-dependent*, i.e., it depends on the state. For example, whether an update of a post is confidential or not in CoSMed intuitively depends on the (dynamic) fact whether an observer is currently a friend of the owner of the post or not.

CoSMed's step function has been formalized in the interactive theorem prover Isabelle/HOL [NPW02]. Executable Scala code has been automatically generated from this specification using Isabelle's code generator [HN10]. This code forms CoSMed's kernel. Around the extracted code, a layer of web-specific code that provides CoSMed's user interface has been implemented, invoking kernel actions in response to user requests [BPPR16, Section 2.2], using the Scalatra web framework.[3]

4.1.2. Extension to a Distributed System

We extended the original system by mechanisms for communicating and sharing data with other nodes located at different sites across the Internet. All nodes have identical behavior, i.e., they are instances of CoSMeDis. Their internal states will be different, however, due to their different local interactions with their respective users and the communication among each other. The resulting system is similar to existing, federated social networks, such as Diaspora. A user signs up at one of the nodes in the network and uses it to post and read content and maintain friendship links. Employing the new inter-node communication features, a user can also establish friendship links to users on other nodes and exchange content across nodes. Public posts are available to all users of connected nodes, whereas private posts are only accessible by local and remote friends of the post owner.

Communication Infrastructure

We consider networks with an arbitrary number of CoSMeDis nodes. A node is designated by a unique ID, its URL. We implement an asymmetric communication model. Any two nodes with IDs nid_1 and nid_2 can agree on a client-server relationship: The client nid_1 makes a request and the server nid_2 approves it (both actions being triggered by the admin users of the corresponding nodes). After that, nid_2 can share its posts with nid_1, and users of nid_2 are allowed to mark selected users of nid_1 as friends. Hence, the admins are responsible for setting up inter-node communication in addition to approving local user creation. From a user perspective, the system looks like CoSMed, except that users can see posts from other nodes if the owner has granted them access, and can add remote friends by selecting a node and entering a username.

To achieve the above, we extend the system's state with communication infrastructure (the IDs of the registered client and server nodes) and shared data (inter-node friendship and shared posts). We also add new types of actions to support the desired communication: sendServerReq and recvClientReq, sendPost and recvPost, and sendUpdRFriend and recvUpdRFriend. They come in pairs: There is an action on the receiving side to match that on the sending side. For successful communication, the parameters of these actions (consisting of user, node and post IDs, post content, etc.) must also match. Here is the intended workflow and the matching patterns for these actions:

[3] `http://scalatra.org/`

Request server connection The admin uid_1 of a node nid_1 can issue a server request to another node nid_2, with the intention of establishing a client-server relationship. The corresponding action is (sendServerReq, uid_1, nid_2, *request*), where *request* is the content of the request message. When the request reaches nid_2, the action (recvClientReq, nid_1, *request*) is triggered on nid_2, to the effect of recording in nid_2's state that nid_1 wishes to become a client.

Connect client with server At a later time, the admin uid_2 of nid_2 can inspect and approve the request. This is done through the communication action (connectClient, uid_2, nid_1), which registers the node nid_1 as a client in nid_2's state. The matching action on nid_1's side is (connectServer, nid_2), which registers the node nid_2 as a server in nid_1's state.

Share posts After nid_1 and nid_2 have recognized each other as a client-server pair, other communication actions are possible. The admin uid_2 of the server nid_2 (or a program running on the admin's behalf) can send a local post *pid* at any time to the client nid_1, via (sendPost, uid_2, nid_1, *pid*). This action produces the output (pid, pst, uid_2', v), consisting of the post ID *pid*, the content *pst* of the post, the ID uid_2' of the post's owner, and information on the post's visibility, v. In this output, *pid* is copied from the action's parameter, whereas all the other components are retrieved from nid_2's state. The matching action on the nid_1 side is (recvPost, nid_2, pid, pst, uid_2', v). Sending an updated version of a previously shared post is possible—it has the effect of updating the remote version. A flag is stored in the server node's state for each shared post with each client, indicating whether the remote version is up to date.

Assign remote friends Post owners retain control on the remote access rights for their friend-only posts: for example, the remote version of *pid* will only be accessible to users uid_1' of nid_1 which uid_2' designates as remote friends. Remote friend designation is done through the action (sendUpdRFriend, uid_2', nid_1, uid_1', *st*), which sends an update to the friendship-like permission from user uid_2' of nid_2 to user uid_1' of nid_1; the flag *st* indicates if friendship is to be *granted* or *revoked*. The matching action on the nid_1 side is (recvUpdRFriend, nid_2, uid_2', uid_1', *st*), which updates the indicated permission.

Implementation

CoSMeDis is implemented similarly to its predecessor, CoSMed. It consists of three layers: the kernel (generated automatically), the API layer, and the outer layer (the last two implemented manually).

The kernel consists of the I/O automaton extracted from the Isabelle specification to Scala code. In addition to regular data (on users, posts and friendship), the kernel state also contains identity checking data: passwords for users and keys for client and server nodes. Moreover, the kernel actions take passwords and/or keys as parameters—omitted here to enhance readability.

The API layer forwards requests back and forth between the I/O automaton kernel and the outside world. It converts the payload of HTTP(S) requests into elements of *Act*, which are passed to the automaton; the output retrieved from the automaton is then converted into JSON output, which is delivered as the API response. Special treatment is given to data that cannot be reasonably stored in memory, namely, the optional image files associated to posts. These are stored on the disk, while the kernel state stores the paths to their locations. They are all placed in a single directory, and the names of the files are the (guaranteed to be non-overlapping) post IDs. When a user requests the read of a post image, the API layer invokes the corresponding reading action from the kernel to retrieve the path to that file—then the file is offered for download.

The outer layer handles the user interface and performs session management. It directs user requests to the API layer and displays the results back to the user or makes remote API requests to other nodes (for communication actions).

The main behavioral differences between CoSMeDis and CoSMed are the following:

- A user action can produce not only local effects, but possibly also a request to a different node. For example, the single action $(\mathsf{sendPost}, uid_2, nid_1, pid)$ triggers a local API call and a remote API call to nid_1.
- Not only users, but other nodes can issue actions as well. For example, $(\mathsf{recvPost}, nid_2, pid, pst, uid_2', v)$ is an action issued by a remote node nid_2.

Making sure that send and receive actions are correctly matched is achieved by a transactional policy. For example, when the admin of node i issues a sendPost request indicating j as the target node, the following happens:

- The sendPost action is run locally, producing the new state σ_i' and the output o_i; but the new state is not yet committed.
- If sendPost was successful, it outputs $o_i = (pid, pst, uid_i', v)$, i.e., the post ID, content, owner, and visibility. This output is converted into a corresponding recvPost request with those parameters (together with the ID of the sending node i) and invoked remotely on node j.
- If node j responds with output OK, the new state σ_i' is committed at node i.

CoSMeDis is delivered as a bundle, which can be deployed at any location on the web to form a new node. The Scala code for the CoSMeDis I/O automaton is significantly larger than that of CoSMed: 2700 compared to 1650 LOC. As expected, the other parts of the application are also larger: 870 vs. 610 for the API layer and 900 vs. 720 for the outer layer.

Modeling Communication

In order to formally model the behavior of a *network* of CoSMeDis nodes, we instantiate the n-ary composition operator introduced in Section 2.3.2. We assume

Table 4.1.: Matching actions and outputs in CoSMeDis

a_i	o_i	a_j	o_j
$(\mathsf{sendServerReq}, u_i, j, request)$	$request$	$(\mathsf{recvClientReq}, i, request)$	OK
$(\mathsf{connectClient}, u_i, j)$	OK	$(\mathsf{connectServer}, i)$	OK
$(\mathsf{sendPost}, u_i, j, pid)$	(pid, pst, u_i', v)	$(\mathsf{recvPost}, i, pid, pst, u_i', v)$	OK
$(\mathsf{sendUpdRFriend}, u_i', j, u_j', st)$	(u_i', u_j', st)	$(\mathsf{recvUpdRFriend}, i, u_i', u_j', st)$	OK

an arbitrary but fixed finite set N of identifiers of the nodes in the network. Each node is modeled as an LTS analogous to the CoSMed automaton described above. A CoSMeDis LTS has output and input events (in the sense of network communication with other nodes, not user input and output). The set of output events Out_i of node i contains successful invocations of the sendServerReq, connectClient, sendPost, or sendUpdRFriend actions. Conversely, the input events In_i contain recvClientReq, connectServer, recvPost, or recvUpdRFriend actions.

For an output and input event to match,

- the types of their actions need to be dual to each other, e.g., sendPost and recvPost,
- the ID of the receiving node must be the target node passed to the output action, and
- the parameters of the input action must match the output of the sending action, e.g., the post content output by sendPost must match the input parameters of recvPost.

Intuitively, what is sent must coincide with what is being received. Formally, $e_i \parallel_{i,j} e_j$ holds iff $i \neq j$ and the actions and outputs of e_i and e_j adhere to the patterns summarized in Table 4.1. Note that, even though we have chosen to include local states in the transitions of CoSMeDis, they are ignored for synchronization between nodes, because we assume that the nodes have no shared state. All information relevant for synchronization is contained in the action parameters and outputs.

4.2. Confidentiality Requirements

We now discuss the confidentiality requirements that had been verified for CoSMed as instances of BD Security, and that we would like to lift to CoSMeDis.

4.2.1. Post Confidentiality

For CoSMed [BPPR16], we had proved confidentiality properties of the form: Only under the circumstances specified by a given policy may users learn information about the system's documents. Our running example in this chapter will be:

> (P_1) A group of users can *learn* nothing about the updates to a post content *beyond* the nonexistence of an update *unless* one of them is the admin or the post's owner, or becomes a friend of the owner, or the post is marked as public.

We assume that the observers are an arbitrary but fixed group of users, denoted UIDs. Their observations consist of the sequence of actions they perform and corresponding outputs they receive.

$$Ev^{obs} \equiv \big\{(\sigma, a, o, \sigma') \mid \mathsf{userOf}(a) \in \mathsf{UIDs}\big\}$$

$$getObs\,\big((\sigma, a, o, \sigma')\big) \equiv (a, o)$$

Note that *failed* actions of the observers are included here, too. This implies that not only regular outputs, but also error messages must not leak confidential information. The secret information consists of the sequence of content updates for the post with a given (arbitrary but fixed) identifier PID. Hence, the set of secret events consists of update transitions for PID, and the secret information in each of those events is the new content.

$$Ev^{sec} \equiv \big\{(\sigma, a, o, \sigma') \mid \exists uid, pst.\ a = (\mathsf{updatePost}, uid, \mathsf{PID}, pst) \wedge o = \mathsf{OK}\big\}$$

$$getSec\,\big((\sigma, (\mathsf{updatePost}, uid, \mathsf{PID}, pst), \mathsf{OK}, \sigma')\big) \equiv pst$$

However, unauthorized observers might learn that no update has been performed yet. In particular, if the system has not been initialized yet, the sequence of updates is definitely empty. We consider this information leak to be harmless and encode it into the bound by specifying that observers may learn about the emptiness of the sequence of updates.

$$B \equiv \big\{(sl, sl') \mid sl = \langle\rangle \longrightarrow sl' = \langle\rangle\big\}$$

Of course, observers may *legitimately* learn about the content of the post if the owner of the post decides to declare it public, or if the owner adds an observer as a friend. Hence, there are certain conditions under which confidential information is about to be declassified to observers. The properties that we have verified for CoSMed and CoSMeDis take into account these conditions in a *dynamic* way, allowing them to be triggered and *released* repeatedly. For example, when the post owner adds an observer as a friend, then this declassifies the current content of the post to them, as well as subsequent updates as long as the friendship persists. However, the owner might decide to *cancel* the friendship with the observers later. The confidentiality

of any updates that occur *afterwards* should be protected again. We specify this by enriching the bound so that it imposes different requirements at different times, depending on whether an observer is currently a friend of the post owner or not. Since this enrichment of the bound with dynamic declassification is largely orthogonal to the question of compositionality, we defer its discussion to Section 4.4.3 and focus on a simplified property first: When the post owner adds an observer as a friend, this simplified property stops making *any* guarantees about the confidentiality of the post, and *everything* is declassified, including past and future updates. We specify this in terms of a declassification *trigger*. Triggers are part of the original notion of BD Security [KLP14] and allow the specification of points in a trace after which the confidentiality guarantee is no longer required to hold. Formally, they are predicates T on transitions. If $T(e)$ holds, then the transition e causes the declassification trigger to fire. For post confidentiality, this happens when the post owner declares the post to be public, or adds an observer as a friend (or the owner is an observer themselves). Moreover, we assume that the administrator has privileged access to the data stored in the system, so we do not guarantee confidentiality if the administrator is an observer. We formalize these conditions in the following trigger.

$$\begin{aligned} T\left((\sigma, a, o, \sigma')\right) \equiv\ & \mathsf{owner}(\sigma', \mathsf{PID}) \in \mathsf{UIDs} \vee \\ & \mathsf{UIDs} \cap \mathsf{friendIDs}(\sigma', \mathsf{owner}(\sigma', \mathsf{PID})) \neq \emptyset \vee \\ & \mathsf{visPost}(\sigma', \mathsf{PID}) = \mathsf{public} \vee \\ & \mathsf{admin}(\sigma') \in \mathsf{UIDs} \end{aligned}$$

Triggered BD Security only makes guarantees for traces in which the trigger does not fire.

Definition 4.1. Let $\mathcal{V}$ be a view on *LTS*, and T: *Ev* $\to$ *Bool* be a predicate on events. *LTS satisfies triggered BD security w.r.t.* $\mathcal{V}$, B, *and* T iff for all $t \in [\![LTS]\!]$ and $s' \in \mathsf{Sec}^*$,

$$(S_{\mathcal{V}}(t), s') \in B \wedge \mathrm{filter}(T, t) = \langle\rangle \longrightarrow (\exists t' \in [\![LTS]\!].\ O_{\mathcal{V}}(t') = O_{\mathcal{V}}(t) \wedge S_{\mathcal{V}}(t') = s')$$

Note that, from a theoretical point of view, triggers add (almost) no expressivity to BD Security, as we show in Appendix D: Triggers can be encoded into the bound. This is why we have not discussed them so far. However, triggers are convenient to keep our example properties simple while we discuss the instantiation of the compositionality result, so we make explicit use of triggers in this chapter.

The above post confidentiality property holds for CoSMed. Lifting it to the distributed system is not entirely trivial, because posts may now be communicated to other nodes in the network. We want to show that the distributed system still respects the policies that users specify for their posts. As in CoSMed, they do this by setting the visibility level of posts to either public or friends-only, and in the latter case, restricting access by maintaining a list of who their friends are. The only difference from a user perspective is that friends may now also include users at other nodes. From a verification perspective, it is no longer sufficient to consider

only one network node; *all* nodes have to protect the confidentiality of posts that are either uploaded by local users or received from other nodes. In Section 4.4, we will discuss how to decompose the global post confidentiality property (P_1) into sufficiently strong properties for individual nodes.

It has to be noted that the scope of our verification is the automaton corresponding to the system kernel, formalized in Isabelle and extracted into Scala, as discussed in Section 4.1. The other components around it are trusted. In particular, we trust the API layer to be a correct translator of HTTP(S) requests to automaton actions and of automaton output to JSON output, and the session management code to correctly keep track of users logged in simultaneously. Extending the scope of verification to these outer layers would require us to step outside the reach of Isabelle. However, compared to the kernel, these layers require the establishment of much simpler, noninterference-like properties: that they transport the data back and forth between the end user and the verified kernel, without doing anything "interesting," like mixing identities. Such properties seem to be in the scope of static information-flow analyzers such as Joana [HS09], or of hybrid verification schemes such as [KTB+15].

4.2.2. Friendship Confidentiality

Another source of information for which we have verified confidentiality guarantees is friendship between users. We consider the friendship information between two arbitrary but fixed users UID_1 and UID_2. Either one of them can request friendship from the other. Such a request may be accompanied by a greeting message, which is kept confidential from all other users. If the request is accepted, the two users are registered as friends in the system. The only information about requests that other users may learn is the *static* knowledge that a request must have been made (and accepted) before a friendship is established. Since the system allows the listing of friends of friends, observers *may* learn about this friendship status, but only if they are themselves friends of UID_1 or UID_2. Again, we formalized this as a property with dynamic trigger. In particular, if UID_1 and UID_2 cancel their friendships with the observers, their friendship status becomes confidential again. We discuss the details in Section 4.4.3.

The confidentiality of local friendship status and requests extends more or less trivially from a single CoSMed system to a whole CoSMeDis network, since this information is never communicated to other nodes—we have implemented the listing of friends only locally, not remotely. However, adding and deleting users of *other* nodes as friends in CoSMeDis *does* involve communication, so we face a similar challenge as for post confidentiality: all involved nodes need to keep remote friendship information confidential, and we need to find local properties of the individual nodes that imply the desired security guarantee of the distributed system. We will discuss these in Section 4.4, but first focus on the compositionality result we use and the requirements it places on the local security properties.

4.3. A Practical n-ary Compositionality Result

In order to formally verify the intended security properties of CoSMeDis, we developed a compositionality framework in the same spirit as the one presented in Chapter 3, but much easier to use. It formulates a set of simplifying assumptions that let us avoid proving more complicated side conditions like flexible scheduling of secrets and observations.

In a nutshell, the first main simplifying assumption is that all communication transitions are treated as observable to some degree. Note that this does not mean that they are fully observable: there may still be secret information contained in them, only some aspects of them have to be observable. In particular, the fact *whether* communication occurs, i.e. the metadata of communication, is considered to be observable. For example, an observer might see that a message has been sent over the network, but the content of the message is kept confidential. In practice, this can be realized by *encrypting* communication. The assumption of observable communication metadata is then in line with a Dolev-Yao style network attacker who can inspect and control communication without breaking encryption. This seems to be a reasonable assumption in many scenarios. Note that strengthening the power of the observer does not weaken our security property. To the contrary, it becomes *stronger*: even if the attacker could inspect most of the network traffic, the system is guaranteed to not leak confidential information via communication, e.g. via the number or scheduling of messages.

A second assumption is that the observations and secrets contain sufficient information to fully determine the synchronization behavior of communication events.

The third main assumption is that only one of the subsystems is primarily responsible for generating secret information. This node is called the *issuer* of the secret. For example, when considering the confidentiality of a given posting uploaded by a user to the social media platform, the issuer is the node of the distributed network where the post has been created. The other nodes may receive secret information from the source, process it locally, and communicate secret information back to the source. However, they are not allowed to generate new secret information independently of the source. These nodes are called the *receivers*.

The last assumption is called *secret polarization* in [BPPR17] in the case of binary composition, and we have incorporated it into the compositionality result of Chapter 3 as one way of solving the challenge of scheduling of secrets and observations (cf. Section 3.3.3). Like the other two assumptions above, it can be formalized purely in terms of the views. Hence, these assumptions are much easier to verify than the more complex side conditions presented in Chapter 3. This simplicity comes at the price of reduced generality, although we believe that the assumptions apply in many scenarios other than the concrete case study. In particular, we believe that the assumptions of observable network traffic and information-rich observations and secrets are quite natural for a compositional approach and should not be too problematic. However, the assumption of a single issuing node limits the scope of the compositionality result presented in this section more significantly. Intuitively,

it only applies to systems where each item of potentially secret information is controlled by one node. This setup is similar to a single-master (or master-slave) model in database replication [SS05], with a designated master node for each secret. In contrast, a multi-master model, where different nodes perform modifications of a secret concurrently and merge them asynchronously, is outside the scope of this simplified setup. Note that it is *not* a problem if there are multiple issuers of *different* independent items of secret information, e.g., different posts. We discuss this in detail in Section 4.3.3. We now proceed to formalize the above assumptions in general terms.

4.3.1. Formalization

In this section, we consider an arbitrary network of LTSs *Net* indexed by a set of identifiers N. The communication interfaces between the nodes are determined by a family $(\|_{i,j})_{i,j\in N}$ of synchronization relations, as introduced in Section 2.3.2. We assume that each network node LTS_i for $i \in N$ satisfies BD Security w.r.t. a local view $\mathcal{V}_i$ and a bound B_i. For the overall network, the canonical view simply lifts local observations and secrets to the network context, tagging them with the identifiers of the node(s) at which they originated, analogous to how we constructed the *events* of the network in Definition 2.5.

Definition 4.2. The *composition of* $(\mathcal{V}_i)_{i\in N}$ *w.r.t.* $(\|_{i,j})_{i,j\in N}$ is the view

$$\mathcal{V} = (Ev^{obs}, getObs, Ev^{sec}, getSec)$$

with

$$\begin{aligned}
Ev^{obs} &= \{(i, e_i) \mid e_i \in Ev_i^{obs} \setminus If_i\} \cup \\
&\quad \{(i, e_i, j, e_j) \mid e_i \in Ev_i^{obs} \wedge e_j \in Ev_j^{obs} \wedge e_i \|_{i,j} e_j\} \\
getObs(e) &= \begin{cases} (i, getObs_i(e_i)) & \text{if } e = (i, e_i) \\ (i, getObs_i(e_i), j, getObs_j(e_j)) & \text{if } e = (i, e_i, j, e_j) \end{cases} \\
Ev^{sec} &= \{(i, e_i) \mid e_i \in Ev_i^{sec} \setminus If_i\} \cup \\
&\quad \{(i, e_i, j, e_j) \mid e_i \in Ev_i^{sec} \wedge e_j \in Ev_j^{sec} \wedge e_i \|_{i,j} e_j\} \\
getSec(e) &= \begin{cases} (i, getSec_i(e_i)) & \text{if } e = (i, e_i) \\ (i, getSec_i(e_i), j, getSec_j(e_j)) & \text{if } e = (i, e_i, j, e_j) \end{cases}
\end{aligned}$$

The translation theorem we discuss in the next subsection will allow us to customize this view and the bound for the network. The main advantage of the canonical view is that composed secrets contain all the information of local secrets. Indeed, we can uniquely restore the local sequences of secrets. Analogous to the decomposition of traces, the projection of the global secret sequence sl to the node i is denoted

$sl \restriction i$ and defined recursively by $\langle\rangle \restriction i = \langle\rangle$ and

$$(s \cdot sl) \restriction i = \begin{cases} s_i \cdot (sl \restriction i) & \text{if } s = (i, s_i) \\ & \text{or } s = (i, s_i, j, s_j) \\ & \text{or } s = (j, s_j, i, s_i) \\ sl \restriction i & \text{otherwise} \end{cases}$$

This allows us to define the canonical bound for the network succinctly as the "synchronized intersection" of the bounds of the nodes.

Definition 4.3. The *bound composition of* $(B_i)_{i \in N}$ *w.r.t.* $(\|_{i,j})_{i,j \in N}$ is the bound

$$B = \left\{(sl, sl') \in \mathsf{Sec}^* \times \mathsf{Sec}^* \mid \forall i \in N.\ (sl \restriction i, sl' \restriction i) \in B_i\right\}$$

where Sec is the range of *getSec*, i.e. secrets of the form (i, s_i) or (i, s_i, j, s_j) produced by secret network events.

We now formalize the assumptions we discussed above to guarantee that the local security guarantees of the nodes can be composed without further side conditions. The first main assumption is observable network traffic between the nodes, in order to avoid issues arising from neutral events at the communication interface (cf. Section 3.3.1).

Definition 4.4. The views $\mathcal{V}_i$ and $\mathcal{V}_j$ have *observable network traffic w.r.t.* $\|_{i,j}$ iff

- $e_i \in \mathit{If}_{i,j}$ implies $e_i \in \mathit{Ev}_i^{obs}$ and
- $e_j \in \mathit{If}_{j,i}$ implies $e_j \in \mathit{Ev}_j^{obs}$.

Moreover, in order to avoid synchronization failures for events with matching secrets and observations (cf. Section 3.3.2), we assume that the (combination of) observations and secrets of communication events contain enough information to capture their synchronization behavior. For this purpose, we define auxiliary predicates that capture matching secrets and observations: two secrets match if they can be produced by a pair of events that synchronize.

$$s_i \parallel_{i,j}^{sec} s_j \longleftrightarrow \left(\exists e_i, e_j.\ e_i \parallel_{i,j} e_j \wedge \mathit{getSec}_i(e_i) = s_i \wedge \mathit{getSec}_j(e_j) = s_j\right)$$

Matching observations $\|_{i,j}^{obs}$ are defined analogously. We assume that matching observations and secrets of events imply that they synchronize.

Definition 4.5. The views $\mathcal{V}_i$ and $\mathcal{V}_j$ are *communication-strong w.r.t.* $\|_{i,j}$ iff, for all $e_i \in \mathit{If}_{i,j}$ and $e_j \in \mathit{If}_{j,i}$, $e_i \parallel_{i,j} e_j$ holds if

- $e_i \in \mathit{Ev}_i^{obs} \longrightarrow \mathit{getObs}_i(e_i) \parallel_{i,j}^{obs} \mathit{getObs}_j(e_j)$ and
- $e_i \in \mathit{Ev}_i^{sec} \longrightarrow \mathit{getSec}_i(e_i) \parallel_{i,j}^{sec} \mathit{getSec}_j(e_j)$.

We also assume that the views are symmetric in the sense that either *both* or *none* of the events in a communicating pair are secret and observable. Recall that the general compositionality result for LTSs in Section 3.4 allows asymmetric views, with events that are secret for one component and neutral for the other. However, this leads to additional side conditions, which we want to avoid in this section.

Definition 4.6. The views $\mathcal{V}_i$ and $\mathcal{V}_j$ are *symmetric w.r.t.* $\|_{i,j}$ iff $e_i \|_{i,j} e_j$ implies

- $e_i \in Ev_i^{obs} \longleftrightarrow e_j \in Ev_j^{obs}$ and
- $e_i \in Ev_i^{sec} \longleftrightarrow e_j \in Ev_j^{sec}$.

Finally, in order to avoid problems due to the scheduling of secrets produced by different nodes (cf. Section 3.3.3), we assume that *one* designated node in the network controls the secret. Formally, other nodes may only produce secret events in *synchronization* with the designated issuer node.

Definition 4.7. A node $k \in N$ is the *unique secret issuer in Net w.r.t.* $(\mathcal{V}_i)_{i\in N}$ *and* $(\|_{i,j})_{i,j\in N}$ iff, for all $i \in N \setminus \{k\}$, $e_i \in Ev_i^{sec}$ implies $e_i \in If_{i,k}$.

These assumptions together are sufficient for compositionality.

Theorem 4.1. *Let Net be a finite set of network nodes indexed by N, whose communication interfaces are determined by $(\|_{i,j})_{i,j\in N}$. Let LTS be the composition of Net w.r.t. $(\|_{i,j})_{i,j\in N}$. Let $(\mathcal{V}_i)_{i\in N}$ be a family of well-defined views on the network nodes. If*

- *each node LTS_i for $i \in N$ satisfies BD Security w.r.t. $\mathcal{V}_i$ and B_i,*
- *the views in $(\mathcal{V}_i)_{i\in N}$ are pairwise symmetric, communication-strong, and have observable network traffic, and*
- *there is a unique secret issuer k in Net w.r.t. $(\mathcal{V}_i)_{i\in N}$ and $(\|_{i,j})_{i,j\in N}$,*

then LTS satisfies BD Security w.r.t. $\mathcal{V}$ and B as defined in Definitions 4.2 and 4.3.

The proof proceeds by an inductive argument, beginning with the issuer node and adding one receiver node at a time in the inductive step. In order to compose a sub-network with an additional node, we use a binary compositionality that makes direct use of the simplifying assumptions [BPPR17, Theorem 1]. The details of that proof can be found in the paper and the accompanying Isabelle formalization. Alternatively, given the framework presented in Chapter 3, we could also use Theorem 3.4 in the inductive step, since the well-behavedness conditions of Lemma 3.3 are implied by the assumptions made here:

- due to observable network traffic, there are no neutral events, so $N_{i,j}$ is empty and the side condition about acceptance of neutral events is trivially satisfied,

- due to communication-strong views, the synchronization behavior of communication events is fully determined by their observations and secrets, so $\bowtie_{i,j}$ is empty, and the side condition about matching secrets and observations is satisfied, and
- due to the assumptions of a unique secret issuer and observable network traffic, the components are secret-polarized: the scheduling of secrets is fully determined by the issuer (or the sub-network containing the issuer in the inductive step).

The result of the theorem is a security property for the network w.r.t. the canonical composition of the views and bounds of the nodes, as introduced above.

4.3.2. Translation of Security Properties

After applying the above compositionality result, we will typically want to translate the obtained security property into a more natural formulation. For example, in the case study of the distributed social media platform, when a user uploads a posting, we want to show that the complete distributed network provides the same security guarantee that the local node where the posting was uploaded provides.[4] For this purpose, we use a general translation theorem that allows us to translate between system specifications and security properties, provided that the original (given) security property is *at least as strong* as the target property. We formalize such a translation in terms of functions that map the states, events, observations, and secrets of one system to those of the other. This is very similar to a bisimulation between systems, but a bit more specific, because it uses a function instead of a relation between states, and a bit more general, because it supports a translation of the labels, i.e., the events, in addition to states. We use the former only for simplicity, but the latter is necessary in many of our examples, allowing us to translate from one representation of events to another.

Definition 4.8. A *translation from LTS_1 to LTS_2* is a pair of functions (π_{St}, π_{Ev}) mapping the states and events, respectively, of LTS_1 to those of LTS_2, such that

- $\pi_{St}(\sigma_1^0) = \sigma_2^0$
- $\sigma_1 \xrightarrow{e_1}_1 \sigma_1'$ implies $\pi_{St}(\sigma_1) \xrightarrow{\pi_{Ev}(e_1)}_2 \pi_{St}(\sigma_1')$
- $\sigma_2 \xrightarrow{e_2}_2 \sigma_2'$ and $\pi_{St}(\sigma_1) = \sigma_2$ imply that there are some e_1, σ_1' such that $\sigma_1 \xrightarrow{e_1}_1 \sigma_1'$, $\pi_{Ev}(e_1) = e_2$ and $\pi_{St}(\sigma_1') = \sigma_2'$

Definition 4.9. A *translation from $\mathcal{V}_1$ to $\mathcal{V}_2$ via (π_{St}, π_{Ev})* is a pair of partial functions (π_O, π_S) mapping the observations and secrets, respectively, of $\mathcal{V}_1$ to those of $\mathcal{V}_2$, such that

[4] If our assumptions hold. In particular, we assume that all nodes in the network run our verified code. If an attacker joins the network as a server operator and runs malicious code instead, our guarantees do not hold any more.

- $\pi_{Ev}(e_1) \in Ev_2^{obs}$ iff $e_1 \in Ev_1^{obs} \wedge getObs_1(e_1) \in \mathrm{dom}(\pi_O)$, and in this case $\pi_O(getObs_1(e_1)) = getObs_2(\pi_{Ev}(e_1))$
- $\pi_{Ev}(e_1) \in Ev_2^{sec}$ iff $e_1 \in Ev_1^{sec} \wedge getSec_1(e_1) \in \mathrm{dom}(\pi_S)$, and in this case $\pi_S(getSec_1(e_1)) = getSec_2(\pi_{Ev}(e_1))$.

Intuitively, each trace of LTS_1 corresponds to one of LTS_2, and each trace of LTS_2 corresponds to at least one (possibly multiple) traces of LTS_1. Hence, LTS_1 is more fine-grained than LTS_2. The same applies to $\mathcal{V}_1$ and $\mathcal{V}_2$. Note that for translating observations and secrets, we use *partial* functions π_O and π_S. This allows us to *weaken* the power of the observer and the notion of secrets by making the sets of observable and secret events smaller. In particular, observable events whose observations are not translated by π_O become *unobservable* in $\mathcal{V}_2$.

Theorem 4.2. *Let (π_{St}, π_{Ev}) be a translation from LTS_1 to LTS_2, and let (π_O, π_S) be a translation from $\mathcal{V}_1$ to $\mathcal{V}_2$ via (π_{St}, π_{Ev}). If*

- *LTS_1 satisfies BD Security w.r.t. $\mathcal{V}_1$ and B_1,*
- *$(sl_2, sl_2') \in B_2$ and $\mathrm{map}_{\pi_S}(sl_1) = sl_2$ imply that there is a sl_1' such that $(sl_1, sl_1') \in B_1$ and $\mathrm{map}_{\pi_S}(sl_1') = sl_2'$,*

then LTS_2 satisfies BD Security w.r.t. $\mathcal{V}_2$ and B_2.

Moreover, if LTS_1 satisfies BD Security with a side condition (P_1, Q_1) w.r.t. $\mathcal{V}_1$ and B_1 (cf. Definition 3.24) and π_{Ev} and π_S are bijections, then LTS_2 satisfies BD Security with the side condition (P_2, Q_2) w.r.t. $\mathcal{V}_2$ and B_2 where

$$P_2(\beta, \alpha, sl') = P_1(\mathrm{map}_{\pi_{Ev}^{-1}}(\beta), \mathrm{map}_{\pi_{Ev}^{-1}}(\alpha), \mathrm{map}_{\pi_S^{-1}}(sl'))$$
$$Q_2(\beta, \alpha, \alpha') = Q_1(\mathrm{map}_{\pi_{Ev}^{-1}}(\beta), \mathrm{map}_{\pi_{Ev}^{-1}}(\alpha), \mathrm{map}_{\pi_{Ev}^{-1}}(\alpha'))$$

Above, $\mathrm{map}_f \colon X^* \to Y^*$ denotes the extension of a partial function $f \colon X \rightharpoonup Y$ to sequences, omitting any elements $x \notin \mathrm{dom}(f)$. In case π_S is defined and injective on all secrets occurring in LTS_1, then it can be inverted and the condition about B_1 and B_2 above simplifies to checking that $(sl_2, sl_2') \in B_2$ implies $(\mathrm{map}_{\pi_S^{-1}}(sl_1'), \mathrm{map}_{\pi_S^{-1}}(sl_1')) \in B_1$.

In combination with Theorem 4.1 we use the translation theorem to show that the network, under a certain condition, is actually secure w.r.t. the declassification bound of the issuer node. We have to prove that the bounds of the receiver nodes are strong enough to support any variation of the secrets required by the bound of the issuer node, or in other words, that the receivers do not declassify any confidential information that has not been declassified already by the issuer node.

Definition 4.10. Let B be the bound composition of $(B_i)_{i \in N}$ w.r.t. $(\|_{i,j})_{i,j \in N}$. The bound B *subsumes the bound of node k* iff

$$\forall i \in N \setminus \{k\}.\ \forall sl \in \mathsf{Sec}^*, sl_k' \in \mathsf{Sec}_k^*.\ (sl \restriction k, sl_k') \in B_k \longrightarrow (\exists sl' \in \mathsf{Sec}^*.\ (sl \restriction i, sl' \restriction i) \in B_i \wedge sl_k' = sl' \restriction k)$$

Theorem 4.3. *Let the preconditions of Theorem 4.1 hold for a network with the unique secret issuer k. If the bound composition B subsumes the bound of node k, then the network is secure w.r.t. B_k and the view*

$$\mathcal{V}' = \left(Ev^{obs}, getObs, Ev^{sec}, getSec'\right)$$

where $getSec'$ extracts the secret of the issuer node k, i.e., $getSec'(e) = getSec_k(e_k)$ if e has the form (k, e_k) or (k, e_k, i, e_i) or (i, e_i, k, e_k).

All of the preconditions of this theorem hold for the properties in our case study, as we will show in Section 4.4.

4.3.3. Combining Independent Secret Sources

The assumption of a unique secret issuer node is a significant limitation of the above network compositionality result. It restricts us to considering the confidentiality of only *one* secret source at a time. In the case study of our social media platform, for example, this could be one given (arbitrary but fixed) posting that a user has uploaded, or the friendship information between two given users. However, a social media platform of course handles many different posts by different users at the same time. A legitimate question is therefore whether and how we can handle *multiple sources of secrets simultaneously*.

For example, consider the confidentiality of two different postings made by users on different nodes of the distributed social media platform, say, posting PID_i in node i and posting PID_j in node j. We can instantiate the results of this section for each post separately (where secret polarization holds) and obtain *two* security properties of the distributed system. In this subsection, we show that we can *combine* these two properties *after* composition, obtaining a property of the distributed system about the confidentiality of the two posts *simultaneously*. This relies on two key assumptions:

1. The secrets are *independent* of each other. Indeed, updates to different posts are completely orthogonal in the system; there is no interference between different posts.

2. The *scheduling* of the different secrets is *not confidential*; e.g., the contents of PID_i and PID_j are considered confidential, but the relative timing of uploads is not.

The first assumption guarantees the soundness of our approach to first consider the secrets in isolation, not having to worry about possible inter-dependencies. The second assumption relaxes the final security property, allowing us to ignore the scheduling of secrets. This rules out the scheduling-related issues that we discussed in Section 3.3. The problem there was that the canonical bound made claims about the possible schedulings of secret events that could not always be guaranteed by the

system. Since the relaxed security property that we formulate in this section ignores scheduling completely, these problems cannot occur.

We formalize the two assumptions above as follows. Let (P_i) and (P_j) denote two BD Security properties of the *same* system, where (P_i) comprises the observation producing function O_i, the secret producing function S_i, and the bound B_i, and analogously for (P_j).

We capture the first assumption by assuming that the secret of one security property is *observable* in the view of the other security property. Intuitively, this guarantees that the different secret sources do not interfere with each other. A variation of one of the secrets does not affect the other. Hence, when we have to prove the possibility of changing the secrets (as BD Security requires), we can do this by changing one secret after the other. Formally, the observation function of one property is assumed to fully determine the secret information of the other, i.e., $\mathsf{O}_i(t) = \mathsf{O}_i(t')$ implies that $\mathsf{S}_j(t) = \mathsf{S}_j(t')$. The same has to hold symmetrically for O_j and S_i. This means that the variation of the secret S_i as required by (P_i) is possible without interfering with the secret information of (P_j): the latter stays fixed.

The second assumption is formalized by a combined secret producing function S that does not have the familiar shape of producing an interleaving of secrets, but it produces a *pair* of secret sequences:

$$\mathsf{S}(t) = (\mathsf{S}_i(t), \mathsf{S}_j(t))$$

This combination captures the content of both secrets, but not their scheduling. Consequently, the combined bound is defined as

$$((sl_i, sl_j), (sl'_i, sl'_j)) \in B \longleftrightarrow (sl_i, sl'_i) \in B_i \wedge (sl_j, sl'_j) \in B_j$$

The combined observation function O is assumed to correspond to an *intersection* of the observations of (P_i) and (P_j), i.e., either $\mathsf{O}_i(t) = \mathsf{O}_i(t')$ or $\mathsf{O}_j(t) = \mathsf{O}_j(t')$ implies $\mathsf{O}(t) = \mathsf{O}(t)$. The proof of the combined security property follows easily from the assumptions: Given a trace t with $\mathsf{S}(t) = (sl_i, sl_j)$ and an alternative secret pair (sl'_i, sl'_j) within B, we first invoke (P_i) to obtain t' with $\mathsf{S}_i(t') = sl'_i$, keeping sl_j constant, and then invoke (P_j) to obtain t'' with $\mathsf{S}_j(t'') = sl'_j$, keeping sl_i constant such that $\mathsf{S}(t'') = (sl'_i, sl'_j)$. The combined observation $\mathsf{O}(t)$ remains unchanged in every step.

This proof technique is applicable to arbitrary security properties, as long as the above assumptions are satisfied (and it is straightforward to lift it from pairs to *tuples* of multiple security properties). In the context of this section, the benefit of this technique is that it gives us a way to "circumvent" the assumption of a unique secret issuer in a certain sense: it allows us to apply the compositionality result presented in this section, provided that (in particular) secret polarization holds for each individual source of secrets, and afterwards combining these secret sources, now assuming that they are independent, but *not* needing secret polarization any more. We have formalized this technique in the interactive theorem prover Isabelle, as part of our formalization of the CoSMeDis case study. We have instantiated it for the

example of combining the confidentiality guarantees for two different posts uploaded by users at different nodes of the network. Many of the proof obligations could be handled by Isabelle automatically. The formalization of the technique itself and a few helper lemmas had to be added manually, but the proofs were straightforward. We give some statistics about the formalization in Section 4.4.4.

4.4. Verification of CoSMeDis

We now apply the compositionality result discussed above to verify confidentiality properties for the relevant sources of information in our case study CoSMeDis: posts and friendship information. Before going into detail, we first give a high-level summary of the verification approach we follow.

4.4.1. Approach

We consider the following setting. We have formulated a security property, such as the post confidentiality of Section 4.2.1, that we want to lift from an individual system to a network of communicating nodes. Theorem 4.1 is applicable if there is a unique node in the network that issues secret information, e.g. the content of a specific post that we want to keep confidential. We proceed as follows.

1. We formulate a security property for the receiver nodes that complements that of the issuer, by analyzing which communication events carry confidential information, and what aspects of that information are declassified by the receiver nodes. For example, CoSMeDis nodes may receive updates to the content of posts over the network. We formalize that they shall declassify the content of the confidential post in question only to authorized users and otherwise keep it confidential.

2. In order to satisfy the preconditions of Theorem 4.1, we strengthen the views so that they
   - have observable network traffic, by letting all communication events produce observations (without revealing the secret),
   - are symmetric, by ensuring that, if a communication event is secret, all matching events are also secret for the other node,
   - are communication-strong, by ensuring that the information contained in secrets and observations of communication events *together* are sufficient to determine their synchronization behavior.

   For example, in the case of post confidentiality, an event receiving an update of the post in question contains secret information for the receiver, so we make the corresponding sending event (partially) secret for the *issuer*, in order to obtain symmetric views.

3. The above changes force us to be very explicit about how secret information is communicated between the nodes, and this has to be reflected in the declassification bounds. For example, the bound for post confidentiality on the issuer node now has to make explicit the relation between locally updating the secret post and sending its content to other nodes: each sending event contains the content that was stored by the last update that happened before. More generally, in this step we adapt the bounds to specify what the attacker learns about the additional details of the secret added in the previous step, and (re-)prove the properties of the nodes.

4. We apply Theorem 4.1 to compose the issuer property with the receiver property of the remaining $n-1$ nodes, obtaining a (canonical) security property of the complete network.

5. We apply Theorems 4.2 and/or 4.3 to customize the property to our desired security guarantee. For example, in the case of post confidentiality, we want the policy that the owner of the post sets at the issuer node (visibility to everybody or limited to a given set of friends) to be respected by *all* nodes in the network. Hence, we apply Theorem 4.3 to formally lift the original security property of the issuer node to the overall network.

In case the unique secret issuer assumption does not hold, we have to resort to the more general compositionality result of Chapter 3. The steps needed to apply it will be similar in principle; the main differences will be in steps 2 and 3. On the one hand, the assumptions on the views in step 2 will be weaker, e.g., allowing asymmetric views. On the other hand, there will be additional side conditions in step 3, e.g., totality w.r.t. neutral events, and reordering of observable and secret events. In the case of the properties we want to verify for CoSMeDis, however, we show that the compositionality result with the simplifying assumptions discussed above *is* applicable. We now discuss the steps of the verification in detail.

4.4.2. Instantiation of the Compositionality Result

Consider again the confidentiality of a post PID created at some arbitrary, but fixed node NID, which we call the "issuer" node of the secret. The security property we verified for CoSMed still holds for CoSMeDis. After all, the new communication facilities with other network nodes do not leak additional information to the *local* users of NID. However, in order for the complete network to be secure, we also need to verify that *other* nodes do not compromise the confidentiality of the post *received* from NID. Hence, we formulate a complementary security property for those nodes, which we call "receivers" of the secret.

> (P_2) A group of users UIDs' of node NID' can *learn* nothing about the updates to the content of the post PID of node NID *beyond* the nonexistence of an update *unless* one of them is the admin of NID', or a remote friend of the owner of the post, or the post is marked as public.

This property mirrors the one for the issuer node. The differences are:

- The secret events are not local updates of post content, but communication events receiving the post PID from NID:

$$Ev_2^{sec} \equiv \{(\sigma, a, o, \sigma') \mid \exists pst, uid, v.\ a = (\mathsf{recvPost}, \mathsf{NID}, \mathsf{PID}, pst, uid, v) \wedge o = \mathsf{OK}\}$$

$$getSec_2\left((\sigma, (\mathsf{recvPost}, \mathsf{NID}, \mathsf{PID}, pst, uid, v), \mathsf{OK}, \sigma')\right) \equiv pst$$

- The trigger conditions refer to remote instead of local versions of friendship, post ownership, and visibility, captured in the state variables remoteFriendIDs, remoteOwner, and remoteVis instead of friendIDs, owner, and visPost.

In this section, we will refer to the resulting view as $\mathcal{V}_2$,[5] while we refer to the corresponding view on the issuer side as $\mathcal{V}_1$ (cf. Section 4.2.1).

Both properties can be proved for CoSMeDis, but the side conditions for compositionality are not yet satisfied. In particular, note that almost all communication events are neutral (except for receiving events for PID), i.e., the issuer property (P_1) and the receiver property (P_2) make almost no guarantees about the communication behavior of CoSMeDis nodes. The general compositionality result of Chapter 3 allows neutral output events, but then requires the corresponding receiving event to be secret and further side conditions to hold. In this chapter, we want to leverage the simplifying assumptions of Section 4.3. Hence, we strengthen the views $\mathcal{V}_1$ and $\mathcal{V}_2$ so that they treat all communication events as observable. However, we cannot make the observers arbitrarily strong: some of the communication events carry secret information, namely those transporting the content of PID. We solve this problem by defining the view so that observers can see the *occurrence* of those communication events, but not the secret *content*.

Amendment 1. $\mathcal{V}_1$ is extended as follows:

- $e \in Ev_1^{obs} \longleftrightarrow \mathsf{userOf}(act(e)) \in \mathsf{UIDs} \vee e \in If_{\mathsf{NID}}$
- $getObs_1\,(\sigma_1, a_1, o_1, \sigma_1') = (\mathsf{purgeA}_{\mathsf{PID}}(a_1), \mathsf{purgeO}_{\mathsf{PID}}(o_1))$

where $\mathsf{purgeA}_{\mathsf{PID}} : Act \to Act$ and $\mathsf{purgeO}_{\mathsf{PID}} : Out \to Out$ purge away from *communicating* actions and their outputs the content of PID's post (which constitutes the secret). The observations of $\mathcal{V}_2$ are extended analogously.

For example, the event $(\sigma, (\mathsf{recvPost}, \mathsf{NID}, \mathsf{PID}, pst, uid, v), \mathsf{OK}, \sigma')$ becomes both partially observable and partially secret: its secret information still is the post content pst, while it now also produces the observation $((\mathsf{recvPost}, \mathsf{NID}, \mathsf{PID}, \bot, uid, v), \mathsf{OK})$, where the secret content is purged and replaced by a dummy value $\bot$. This

[5] Formally, $\mathcal{V}_2$ is parametric in the set of observers UIDs' for a given node NID', but we omit this parameter in the text for brevity. In the Isabelle formalization, the security properties of issuer and receiver nodes are specified and verified inside locales with the relevant parameters NID, PID, etc.

models the assumption that the occurrence and non-confidential content of communication is visible to attacker, while confidential content is protected, e.g. by encryption. Consequently, we verify that, even if we allow the attacker to observe almost anything about network communication except the secret itself, the system still does not leak the content of PID. The attacker does, however, learn the number of messages containing PID from observing their occurrence. We adapt the bound of receiving nodes as follows.[6]

Amendment 2. The bound of (P_2) is adapted to

$$(sl, sl') \in B_2 \text{ iff } \text{length}(sl) = \text{length}(sl')$$

For the issuer, we need further amendments. In particular, the issuer and receiver views are not symmetric yet: The recvPost events for PID are (partially) secret for the receivers, but the corresponding sendPost events of the issuer are not. This also means that the views are not communication-strong, since the observation produced by the sendPost event does not contain sufficient information for synchronization (the content of the post is purged away). We solve this problem by treating sendPost events as secret for the issuer.

Amendment 3. (P_1)'s view is extended as highlighted below:

- Secrets are now post contents annotated with the labels upd or snd_i, in order to distinguish between posts produced by an update action, (upd, pst), and posts produced by a sending action to node i, (snd_i, pst).

$$\mathsf{Sec}_1 = (\{\mathsf{upd}\} \times \mathsf{Post}) \cup \bigcup_{i \in N} (\{\mathsf{snd}_i\} \times \mathsf{Post})$$

- $(\sigma, a, o, \sigma') \in Ev_1^{sec}$ iff either
  - $\exists uid, pst.\ a = (\mathsf{updatePost}, uid, \mathsf{PID}, pst) \wedge o = \mathsf{OK}$ or
  - $\exists uid, nid, pst, uid', v.\ a = (\mathsf{sendPost}, uid, nid, \mathsf{PID}) \wedge o = (\mathsf{PID}, pst, uid', v)$
- $getSec_1$ now extracts the content pst from both update actions and outputs for send actions:

$$getSec_1(\_, (\mathsf{updatePost}, \_, \_, pst), \_, \_) = (\mathsf{upd}, pst)$$
$$getSec_1(\_, (\mathsf{sendPost}, \_, i, \_), (\_, pst, \_, \_), \_) = (\mathsf{snd}_i, pst)$$

We now have to specify what the strengthened observers learn about the sending events of PID: the number of messages sent (for each receiver node), and the fact that

[6] This is a detail that, due to lack of space, was omitted from the main text of [BPPR17], but it is included in its appendix, and of course in the Isabelle formalization; otherwise, the proofs would not have succeeded.

the content of a sendPost event corresponds to the current version of the post, i.e., the most recent update. For example, from the sequence of updates and messages

$$\langle(\mathsf{upd}, pst_1), (\mathsf{upd}, pst_2), (\mathsf{snd}_i, pst_3), (\mathsf{upd}, pst_4), (\mathsf{snd}_i, pst_5)\rangle$$

the observers would not learn the confidential contents pst_i, but they would learn that $pst_2 = pst_3$ and $pst_4 = pst_5$, i.e., what is sent coincides with what was last updated. We incorporate this (static) knowledge into the bound as well.

Amendment 4. (P_1)'s bound is extended as follows:

$$(sl, sl') \in B_1 \text{ iff } \big(sl_{\mathsf{upd}} = \langle\rangle \longrightarrow sl'_{\mathsf{upd}} = \langle\rangle\big) \wedge \big(\forall i \in N.\ \mathrm{length}(sl_{\mathsf{snd}_i}) = \mathrm{length}(sl'_{\mathsf{snd}_i})\big) \wedge \mathsf{corr}(sl')$$

where sl_{upd} and sl_{snd_i} denote the filtering of sl for upd and snd_i values, respectively, and $\mathsf{corr}(sl')$ states that updates to (upd) and sending of (snd) posts in sl' is correlated in the above sense.

After these amendments, the security properties still hold for CoSMeDis. The extension of the proofs from CoSMed to CoSMeDis was straightforward, albeit tedious: the additional details in the views and bounds, e.g., the correlation between updates and sending of post content, needs to be accounted for in the proofs. We present some statistics about the effort involved in Section 4.4.4.

What we have gained by amending the properties is that the simplifying assumptions for compositionality we discussed above now hold. We have observable network traffic and views that are symmetric and communication-strong: For non-confidential communication events, the observations contain the complete actions and outputs, which fully determines the synchronization behavior of the events. For confidential communication events, i.e., sendPost and recvPost for PID, the observations contain almost complete actions and outputs, where only the content of PID is purged away. The latter is contained in the secrets, so the *combination* of observations and secrets fully determines the synchronization-relevant information. Moreover, the views are secret-polarized: in the receiver views, only communication events with the issuer are secret.

The preconditions of Theorem 4.1 are therefore satisfied. One thing we need to add to the theorem (as it was presented in this chapter) is support for triggers. As the composed trigger for a network of nodes, we simply take the disjunction of the local triggers: As soon as one node stops guaranteeing confidentiality, the whole network stops providing guarantees. In the post confidentiality example, the local trigger at the issuer node fires when the owner of the post adds a local observer as a friend, or if a local observer is the admin. On the receiver nodes, the local trigger fires when a local observer is added as a *remote* friend by the post owner, or if a local observer is the admin. The global trigger fires when *any* of this happens. Formally, given a family of triggers $(T_i)_{i \in N}$ for the nodes of a network indexed by N, the composed trigger is defined as

$$T(e) = \begin{cases} T_i(e_i) & \text{if } e = (i, e_i) \\ T_i(e_i) \vee T_j(e_j) & \text{if } e = (i, e_i, j, e_j) \end{cases}$$

As part of our Isabelle formalization, we have verified that Theorem 3 of [BPPR17] holds, which corresponds to Theorem 4.1 with triggers added in this way.

We apply the theorem to a network of n CoSMeDis nodes with issuer node k by plugging in the extended $\mathcal{V}_1$, B_1, and T_1 for node k, and $\mathcal{V}_2$, B_2, and T_2 for the remaining $n-1$ nodes. Note that Theorem 4.1 does not make assumptions about the number of nodes or the topology of the network, only about the security properties. We obtain a property for the network that is a combination of the properties of the nodes. The observers are sets of local users at each node, together with a network adversary who can inspect communication between the nodes. The secrets are local updates to PID at the issuer node, as well as messages from the issuer to receivers containing PID. The bound is the "synchronized intersection" of the bounds of the nodes, as introduced above. In fact, it turns out to correspond to the bound of the issuer. Note that the receiver bounds mirror that of the issuer: unless the trigger occurs, the receivers only declassify the number of messages containing PID received from the issuer. This is declassified already by the issuer. Hence, the precondition of Theorem 4.3 is satisfied, and we obtain the security of the overall network w.r.t. the secrets of the issuer and its bound B_1.

These secrets still include the communication of the confidential post to other nodes, correlated to the local updates of the post content. The security property we originally formulated for CoSMed only considered local post updates as secret and local users as observers. A corresponding property for CoSMeDis is the following.

> (P') A coalition of n groups of users, UIDs_i for each node i, can learn nothing about the updates to PID's content at node k *beyond* the nonexistence of an update *unless* one of the following holds:
>
> 1. one of UIDs_k is the admin, or is PID's owner, or becomes a friend of the owner, or PID is marked as public
> 2. the post is being shared by node k with some node $i \neq k$ and either one of the users in UIDs_i is the admin or becomes a remote friend of PID's owner, or PID is marked as public.

The bound is the same as for CoSMed, but the trigger is the composed trigger we discussed above. Indeed, this property holds for CoSMeDis. We derive this by applying the translation Theorem 4.2 to the security property we obtained from the compositionality Theorem 4.3. We weaken the observers by dropping the network adversary and only considering local users, and we weaken the secrets by dropping communication of post content and only considering local updates. The bound therefore does not have to talk about the number of messages or the correlation between updates and sending of posts any more, and it simplifies to the original

CoSMed bound with $(sl, sl') \in B$ iff $(sl = \langle\rangle \longrightarrow sl' = \langle\rangle)$. The trigger does not change: It is still the composed trigger we obtained above.[7] Formally, we instantiate Theorem 4.2 by choosing the following parameters. The state and event translation functions π_{St} and π_{Ev} are the identity, since the system model does not change. The observation translation function π_O is also the identity, but only on *local* observations, i.e., actions of local users. For communication events, it is not defined, so these events become unobservable. The secret translation function π_S is also defined only on local (update) secrets, and it extracts the content of the post:

$$\pi_S\left((\mathsf{upd}, pst)\right) = pst$$

Note that this function is a partial bijection. The preconditions of Theorem 4.2 are easy to check, and we obtain that (P') holds for the CoSMeDis network.

4.4.3. Verified Properties

The properties we verified for the original CoSMed are more fine-grained than the one we discussed so far. They factor in *temporal* aspects of declassification by considering *dynamic* trigger conditions that allow declassification only during certain phases of system execution. We summarize these properties for post content and friendship information of CoSMed in Table 4.2. In the rest of this subsection, we discuss how we modeled them formally, and how we applied our compositionality heuristic to lift them to CoSMeDis. The lifted properties are summarized in Table 4.3.

Post Confidentiality

The above version of post confidentiality with static trigger declassifies the content as soon as the trigger conditions hold at some point during the trace. This is unsatisfactory in many situations. For example, consider a user of the system becoming a friend of an observer, then canceling that friendship again, and only later creating the post PID. The property we discussed so far does *not* protect the confidentiality of PID in this trace at all, because the trigger had fired once, *before* the secret is produced. A more reasonable property would only declassify secrets *during* phases in which the trigger holds, and start protecting their confidentiality again when the trigger is released. For CoSMed's posts, we have proved the following property.

> A coalition of users UIDs can learn nothing about the updates to PID's content *beyond* those updates that are performed *while* one of the following holds *or the last* update *before* one of the following starts to hold:
>
> - a user in UIDs is the admin, or
> - a user in UIDs is the post's owner or a friend of the post's owner, or
> - the post is marked as public.

[7] The version of Theorem 4.2 that we formalized in Isabelle also supports weakening a security property by adding conditions under which the trigger fires. Formally, a trigger T_1 can be translated to T_2 if $T_1(e)$ implies $T_2(\pi_{Ev}(e))$.

Table 4.2.: Confidentiality properties of the original CoSMed (where the observers are a set of users UIDs)

Secret	Bound
Content of a post PID	Updates performed while or last before one of the following holds: • Some user in UIDs is the admin, is the post owner or is friend with the post owner • PID is marked as public
Friendship status between UID_1 and UID_2	Status changes performed while or last before the following holds: • Some user in UIDs is the admin or is friend with UID_1 or UID_2
Friendship requests between UID_1 and UID_2	Existence of accepted requests while or last before the following holds: • Some user in UIDs is the admin or is friend with UID_1 or UID_2

Table 4.3.: Confidentiality properties lifted from CoSMed to CoSMeDis (with *sets* of observers UIDs_i at each node i)

Secret	Bound
Content of a post PID at node i	Updates performed while or last before one of the following holds: • Some user in UIDs_i is admin of node i, is PID's owner or is friend of PID's owner • PID is marked as public • Some user in UIDs_j for $j \neq i$ is admin of node j or remote friend of PID's owner
Friendship status between local users UID_1 and UID_2 at node i	Status changes performed while or last before the following holds: • Some user in UIDs_i is the admin or is friend with UID_1 or UID_2
Friendship requests between local users UID_1 and UID_2 at node i	Existence of accepted requests while or last before the following holds: • Some user in UIDs_i is the admin or is friend with UID_1 or UID_2

This specifies that, when one of the trigger conditions starts to hold, the observers may learn the current version of the post (corresponding to the last update before the trigger fired) and any updates to it until the trigger is released. After release of the trigger, the observers learn nothing more about further updates (until the trigger fires again, and so on). We formalize this by incorporating the dynamic trigger into the bound. We distinguish two phases: while the trigger holds, we say that the "access window" to the confidential information is *open*, and we say it is *closed* while the trigger does not hold. We enrich the secrets by markers indicating changes in the phase: $(\mathsf{osec}, \mathsf{True})$ indicates that the window has just opened, because the trigger fired, while $(\mathsf{osec}, \mathsf{False})$ indicates that the trigger has been released and the window is now closed. The set of secret events Ev^{sec} is enriched by all transitions (σ, a, o, σ') that cause a phase transition. Formally, these are the events for which $\mathsf{open}(\sigma') \neq \mathsf{open}(\sigma)$ holds, where

$$\begin{aligned}\mathsf{open}(\sigma) \equiv\ & \mathsf{owner}(\sigma, \mathsf{PID}) \in \mathsf{UIDs} \vee \\ & \mathsf{UIDs} \cap \mathsf{friendIDs}(\sigma, \mathsf{owner}(\sigma, \mathsf{PID})) \neq \emptyset \vee \\ & \mathsf{visPost}(\sigma, \mathsf{PID}) = \mathsf{public} \vee \\ & \mathsf{admin}(\sigma) \in \mathsf{UIDs}\end{aligned}$$

captures the trigger conditions. Intuitively, this (potentially) includes the observer becoming the administrator, friendships being established between observers and the post owner, or the visibility of the post being toggled between public and friends-only. Note that our design decision to include states in the transitions allows us to conveniently specify history-dependent trigger conditions. For example, after the post owner has added several observers as friends, the trigger is only released after the post owner has unfriended *all* observers again, i.e. after a sequence of friend deletion actions that depends on the sequence of previous friend addition actions. This is captured implicitly in the above state-based trigger condition.

We extend the secret-producing function to record phase changes as osec values.

$$\begin{aligned}getSec\left((\sigma, (\mathsf{updatePost}, uid, \mathsf{PID}, pst), \mathsf{OK}, \sigma')\right) &\equiv \mathsf{psec}\ pst \\ getSec\left((\_, \_, \_, \sigma')\right) &\equiv \mathsf{osec}\left(\mathsf{open}(\sigma')\right)\end{aligned}$$

This allows us to talk about phase changes in the bound for CoSMed. Intuitively, we define two bounds, one for each phase, and then connect them. For the closed phase, we use essentially our original bound that only declassifies the (non-)existence of an update in that phase. For the open phase, we use the identity as the bound, declassifying everything about the updates during that phase. Upon *transition* from closed to open phase, e.g., when an observer has just become a friend of the post owner, we additionally declassify the *current* version of the post (if any), i.e., the last update during the closed phase. Formally, we define the two bounds using mutual induction, with one bound "switching" over to the other when an osec value occurs. The definition is given in Figure 4.1, where BC and BO are the bounds for the closed and open phase, respectively. Initially, the access window is closed, so we

$$\frac{pstl \neq \langle\rangle \rightarrow pstl' \neq \langle\rangle}{(\mathrm{map}(\mathsf{psec}, pstl),\ \mathrm{map}(\mathsf{psec}, pstl')) \in BC} \quad (1)$$

$$\frac{}{(\mathrm{map}(\mathsf{psec}, pstl),\ \mathrm{map}(\mathsf{psec}, pstl)) \in BO} \quad (2)$$

$$\frac{(sl, sl') \in BO \quad pstl \neq \langle\rangle \longleftrightarrow pstl' \neq \langle\rangle \quad pstl \neq \langle\rangle \longrightarrow \mathrm{last}(pstl) = \mathrm{last}(pstl')}{(\mathrm{map}(\mathsf{psec}, pstl) \cdot (\mathsf{osec}, \mathsf{True}) \cdot sl,\ \mathrm{map}(\mathsf{psec}, pstl') \cdot (\mathsf{osec}, \mathsf{True}) \cdot sl') \in BC} \quad (3)$$

$$\frac{(sl,\ sl') \in BC}{(\mathrm{map}(\mathsf{psec}, pstl) \cdot (\mathsf{osec}, \mathsf{False}) \cdot sl,\ \mathrm{map}(\mathsf{psec}, pstl) \cdot (\mathsf{osec}, \mathsf{False}) \cdot sl') \in BO} \quad (4)$$

Figure 4.1.: The bound for post text confidentiality

take BC to be the overall bound. For a more in-depth discussion of the bound, as well as a verification approach using unwinding, we refer to [BPPR16].

The main question in this chapter is how to lift this property to CoSMeDis. It turns out that the amendments to CoSMed's view and bound for incorporating dynamic triggers are *orthogonal* to the amendments for compositionality we discussed in the previous sections. Recall that we enriched post update secrets to distinguish local updates (upd) and communication of post content to other nodes (snd_i). We simply wrap the openness-enriched secrets of CoSMed into the local updates upd and get three kinds of secrets:

- $(\mathsf{upd}, (\mathsf{osec}, o))$, representing a local change in the trigger status to o,
- $(\mathsf{upd}, (\mathsf{psec}, pst))$, representing a local update of the post content pst, and
- (snd_i, pst), representing the communication of a post content pst to node i.

As the bound for the issuer, we take CoSMed's bound of Figure 4.1 (wrapped into upd), and add Amendment 4 of Section 4.4.2, specifying that the sending of post contents to other nodes is correlated with local updates. This combines the amendments related to dynamic triggers with those related to communication. For the receiver nodes, we have kept the static triggers in the Isabelle formalization; adding dynamic triggers would be straightforward, but require formalization effort without gaining novel insight. Hence, the bound for receivers simply states that only the *number* of communicated updates is declassified, and nothing about their content, *unless* the trigger occurs, e.g., an observer becomes a remote friend of the post owner. Finally, we applied the composition and translation theorems presented in this chapter to derive a security property for CoSMeDis, specifying that the property lifted from CoSMed still holds (albeit with a partially static trigger, due to our simplification of the receivers). The end result for post confidentiality is summarized in Table 4.4.

Table 4.4.: Post confidentiality

Scope	Parameters of security model
CoSMed	$\mathsf{Sec} = \mathsf{psec}\ \mathsf{Post} + \mathsf{osec}\ \mathsf{Bool}$ $(\sigma, a, o, \sigma') \in Ev^{sec}$ iff either 1. $(o = \mathsf{OK} \wedge \exists uid, pst.\ a = (\mathsf{updatePost}, uid, \mathsf{PID}, pst))$ or 2. $\mathsf{open}(\sigma') \neq \mathsf{open}(\sigma)$ with open as defined on page 93 $getSec(\sigma, a, o, \sigma') = (\mathsf{psec}\ pst)$ in case 1 and $(\mathsf{osec}\ \mathsf{open}(\sigma))$ in case 2 $(\sigma, a, o, \sigma') \in Ev^{obs}$ iff $\mathsf{userOf}(a) \in \mathsf{UIDs}$ $getObs\,(\sigma, a, o, \sigma') = (a, o)$ B is defined as in Figure 4.1 and T is vacuously False
CoSMeDis (issuer k)	$\mathsf{Sec}_k = \mathsf{upd}\,(\mathsf{psec}\ \mathsf{Post} + \mathsf{osec}\ \mathsf{Bool}) + \mathsf{snd}_i\ \mathsf{Post}$ $(\sigma, a, o, \sigma') \in Ev_k^{sec}$ iff either 1. $(o = \mathsf{OK} \wedge \exists uid, pst.\ a = (\mathsf{updatePost}, uid, \mathsf{PID}, pst))$ or 2. $\mathsf{open}(\sigma') \neq \mathsf{open}(\sigma)$ with open as defined on page 93 3. $(\exists u, pst, u', v, i.\ a = (\mathsf{sendPost}, u, i, \mathsf{PID}) \wedge o = (pst, u', v))$ $getSec_k(\sigma, a, o, \sigma') = \begin{cases} (\mathsf{upd}, (\mathsf{psec}\ pst)) \text{ in case 1} \\ (\mathsf{upd}, (\mathsf{osec}\ \mathsf{open}(\sigma))) \text{ in case 2} \\ (\mathsf{snd}_i, pst) \text{ in case 3} \end{cases}$ $(\sigma, a, o, \sigma') \in Ev_k^{obs}$ iff $\mathsf{userOf}(a) \in \mathsf{UIDs}_k \vee (\sigma, a, o, \sigma') \in If_k$ $getObs_k\,((\sigma, a, o, \sigma')) = (\mathsf{purgeA}_{\mathsf{PID}}(a), \mathsf{purgeO}_{\mathsf{PID}}(o))$ $(sl, sl') \in B_k$ iff both 1. $(sl_{\mathsf{upd}}, sl'_{\mathsf{upd}}) \in B$ where B is CoSMed's bound and sl_{upd} selects only upd values from sl and extracts their content, and 2. $\big(\forall i \in N.\ \mathrm{length}(sl_{\mathsf{snd}_i}) = \mathrm{length}(sl'_{\mathsf{snd}_i})\big) \wedge \mathsf{corr}(sl')$ T_k is vacuously False
CoSMeDis (receiver i)	$(\sigma, a, o, \sigma') \in Ev_i^{sec}$ iff $\exists pst, u, v.\ a = (\mathsf{recvPost}, k, \mathsf{PID}, pst, u, v)$ and $o = \mathsf{OK}$, in which case $getSec_i(\sigma, a, o, \sigma') = pst$ $(\sigma, a, o, \sigma') \in Ev_i^{obs}$ iff $\mathsf{userOf}(a) \in \mathsf{UIDs} \vee (\sigma, a, o, \sigma') \in If_{i,k}$ $getObs_i\,((\sigma, a, o, \sigma')) = (\mathsf{purgeA}_{\mathsf{PID}}(a), \mathsf{purgeO}_{\mathsf{PID}}(o))$ $(sl, sl') \in B_i$ iff $\mathrm{length}(sl) = \mathrm{length}(sl')$ $T_i\,(\sigma, a, o, \sigma')$ iff either • $\mathsf{admin}(\sigma') \in \mathsf{UIDs}_i$, • $\mathsf{remoteVis}(\sigma', k, \mathsf{PID}) = \mathsf{public}$, or • $\mathsf{UIDs}_i \cap \mathsf{remoteFriendIDs}(\sigma', k, \mathsf{remoteOwner}(\sigma', k, \mathsf{PID})) \neq \emptyset$
CoSMeDis (network)	Ev^{obs} and $getObs$ are composed canonically as in Definition 4.2 T is composed canonically as defined on page 90 $Ev^{sec} = \{(k, e_k)\}$ and $getSec\,(k, e_k) = getSec_k(e_k)$ $B = B_k$

Friendship Confidentiality

In addition to posts, another type of confidential information is that about friendship between users: who is a friend of whom, and who has requested friendship with whom. For CoSMed, we had verified the confidentiality of the (local) friendship information of two arbitrary but fixed users UID1 and UID2 who are *not* in the coalition of observers:

> A coalition of users UIDs can learn nothing about the updates to the friendship status between two users UID1 and UID2 beyond those updates that are performed while a member of the coalition is a friend of UID1 or UID2, or the last update before there is a member of the coalition who becomes a friend of UID1 or UID2.
>
> A coalition of users UIDs can learn nothing about the friendship requests between two users UID1 and UID2 beyond the existence of a request before each successful friendship establishment.

Formally, we declare the access window to friendship information to be open when either an observer is a friend of UID1 or UID2 (since the listing of friends of local friends is allowed), or the two users have not been created yet (since observers know statically that there is no friendship if the users do not exist yet).

$$\begin{aligned}\mathsf{open}_F\ \sigma \quad \equiv \quad & (\exists \mathit{uid} \in \mathsf{UIDs}.\ \mathit{uid} \in \mathsf{friendIDs}(\sigma, \mathsf{UID1}) \vee \mathit{uid} \in \mathsf{friendIDs}(\sigma, \mathsf{UID2})) \vee \\ & \mathsf{UID1} \notin \mathsf{UIDs}(\sigma) \ \vee \ \mathsf{UID2} \notin \mathsf{UIDs}(\sigma)\end{aligned}$$

The relevant transitions are the creation of users and the creation and deletion of friends or friend requests. The creation and deletion of friendship between UID1 and UID2 produces an $(\mathsf{FSec}, \mathsf{True})$ or $(\mathsf{FSec}, \mathsf{False})$ secret, respectively. In the case of openness changes, osec is produced just as for the post confidentiality. Moreover, for the confidentiality of friendship requests, we let the creation of a request between UID1 and UID2 produce $\mathsf{FRSec}(\mathit{uid}, \mathit{txt})$ secrets, where *uid* indicates the user that has placed the request, and *txt* is the request message.

The main inductive definition of the two phases of the declassification bounds for friendship is given in Figure 4.2, where *fs* ranges over friendship statuses, i.e., Booleans. Note that it follows the same "while"-"last update before" scheme as Figure 4.1 for the post confidentiality, but with FSec instead of psec. The overall bound is then taken to be BO_F (since we start in the open phase where UID1 and UID2 do not exist yet) plus a predicate on the secrets that captures the static knowledge of the observers: that the FSec's form an *alternating* sequence of "friending" and "unfriending."

For friendship requests, we additionally require that at least one FRSec and at most two FRSec secrets from different users have to occur before each $(\mathsf{FSec}, \mathsf{True})$ secret. Beyond that, we require nothing about the request values. Hence, the bound for friendship requests states that observers learn nothing about the requests

$$(\mathrm{map}(\mathsf{FSec}, fs),\ \mathrm{map}(\mathsf{FSec}, fs)) \in BO_F \quad (1)$$

$$(\mathrm{map}(\mathsf{FSec}, fs),\ \mathrm{map}(\mathsf{FSec}, fs')) \in BC_F \quad (2)$$

$$\frac{(sl,\ sl') \in BO_F \qquad fs \neq \langle\rangle \iff fs' \neq \langle\rangle \qquad fs \neq \langle\rightarrow\rangle \mathrm{last}\ fs = \mathrm{last}\ fs'}{(\mathrm{map}(\mathsf{FSec}, fs) \cdot (\mathsf{osec}, \mathsf{True}) \cdot sl,\ \mathrm{map}(\mathsf{FSec}, fs') \cdot (\mathsf{osec}, \mathsf{True}) \cdot sl') \in BC_F} \quad (3)$$

$$\frac{(sl,\ sl') \in BC_F}{(\mathrm{map}(\mathsf{FSec}, fs) \cdot (\mathsf{osec}, \mathsf{False}) \cdot sl,\ \mathrm{map}(\mathsf{FSec}, fs) \cdot (\mathsf{osec}, \mathsf{False}) \cdot sl') \in BO_F} \quad (4)$$

Figure 4.2.: The bound on friendship status secrets

between UID1 and UID2 beyond the existence of a request before each successful friendship establishment. In particular, they learn nothing about the "orientation" of the requests (i.e., which of the two involved users has placed a given request) and the contents of the request messages.

These properties of CoSMed trivially hold for CoSMeDis, as well, for the local friendship between UID1 and UID2 on an arbitrary but fixed node k. Indeed, this information is never communicated to any other node, since we have not implemented the remote listing of friends. Hence, the other nodes do not process any secret and are trivially secure. The compositionality theorem therefore allows us to directly lift the security guarantee of k to the whole network.

However, CoSMeDis also allows *remote* friendship links between nodes, which does involve communication. Hence, we prove an additional property for CoSMeDis, summarized in Table 4.5. We consider the remote friends of an arbitrary, but fixed user UID of node k, who is not an observer. Since we assume communication traffic to be observable, we can't keep secret that a remote friendship action occurred and which node it targeted, but we assume that the secret content is unobservable, namely the *type* of remote friendship action (addition or deletion, formalized as a Boolean flag), as well as *who* was added or deleted as a remote friend. Indeed, we verify that this information is kept secret in CoSMeDis from observers who are not friends of UID. However, from observing the occurrence of remote friendship actions and combining it with the static knowledge that addition and deletion of any given remote friend can only occur alternatingly (first addition, then deletion, then addition again, and so on), an observer can deduce the *existence* of remote friends of UID on any given node other than k. The overall property we verify is

> A coalition of n groups of users, UIDs_i for each node i, can learn nothing about the remote friends of the user UID of node k beyond the existence of remote friends on any given node j unless one of UIDs_k is a friend of UID.

Table 4.5.: Remote friendship confidentiality

Scope	Parameters of security model
Issuer k	$\mathsf{Sec}_k = \mathsf{NodeID} \times \mathsf{UserID} \times \mathsf{Bool}$ $(\sigma, a, o, \sigma') \in Ev_k^{sec}$ iff $(\exists u, i, st.\ a = (\mathsf{sendUpdRFriend}, \mathsf{UID}, i, u, st) \wedge u \notin \mathsf{UIDs}_i)$ and $o = \mathsf{OK}$, in which case $getSec_k(\sigma, a, o, \sigma') = (i, u, st)$ $(\sigma, a, o, \sigma') \in Ev_k^{obs}$ iff $\mathsf{userOf}(a) \in \mathsf{UIDs}_k \vee (\sigma, a, o, \sigma') \in If_k$ $getObs_k((\sigma, a, o, \sigma')) = (\mathsf{purgeA}_{\mathsf{UID}}(a), \mathsf{purgeO}_{\mathsf{UID}}(o))$ $(sl, sl') \in B_k$ iff $(sl, sl') \in BC \wedge \mathsf{alter}(sl')$, where • $\mathsf{alter}(sl')$ states that friendship creation and deletion occurs alternatingly in sl for each remote user, and • BC is defined inductively by 1. $(\langle\rangle, \langle\rangle) \in BC$ 2. $((i, u, st) \cdot sl, (i, u', st') \cdot sl')$ iff $u' \notin \mathsf{UIDs}_i \wedge (sl, sl') \in BC$ $T_k(\sigma, a, o, \sigma') = (\exists u \in \mathsf{UIDs}_k.\ u \in \mathsf{friendIDs}(\sigma', \mathsf{UID}))$
Receiver i	Same as for the issuer, only 1. replacing sendUpdRFriend actions by recvUpdRFriend actions coming from the issuer k in the definition of Ev_k^{sec} and $getSec_k$, and 2. defining T_i as vacuously False.
Network	Analogous to the network setup for post confidentiality in Table 4.4, but using the trigger of the issuer for the network instead of those of the receivers.

Formally, the bound given in Table 4.5 states that it must be possible to *replace* the parameters of remote friendship actions arbitrarily (preserving alternation of addition and deletion). The bounds for issuer and receiver nodes are symmetric, because secrets are only generated during communication. Finally, the trigger makes explicit that the friend list of UID is legitimately declassified to local friends of UID (we have only implemented the listing of friends locally, but not remotely, in CoSMeDis).

4.4.4. Isabelle Formalization

As discussed in Section 4.1.2, the functionality of CoSMeDis nodes has been specified in Isabelle as a particular automaton, extending the previously formalized CoSMed automaton. Executable Scala code generated by Isabelle's code extraction formed the basis of the implementation of CoSMeDis.

Our verification focused on the network formed of copies of this I/O automaton. First, we formalized the abstract theorems of Section 4.3. The formalization builds upon a binary compositionality result, similar to the general result of Chapter 3, but making use of the simplifying assumptions presented in this chapter. For proving it,

we need to construct an alternative composed trace from two component traces t_1 and t_2 whose observations and secrets have already been composed, similar to the situation depicted in Figure 3.3. The simplifying assumptions of communication-strength, symmetry of views, and a unique secret issuer guarantee that the traces can always be merged without the need of adapting them (which was necessary for the more general theorem). The proof proceeds by induction on the length of the traces, and is guided by a series of case distinctions on whether the current observations and secrets of the composed system were produced locally by a node or by communication, on whether each of them is secret or observable, etc. The n-ary compositionality Theorem 4.1 is proved inductively by iterating the binary result, composing the issuer node with the other nodes of the network, one at a time. The main difficulty with the translation Theorem 4.2 was its formulation as sufficiently expressive to capture our cases of interest. Once properly formulated, it followed by a straightforward manipulation of the quantifiers in the definition of BD security.

To facilitate the instantiation of these theorems, we proved them within Isabelle modules called *locales* [KWP99], which allow for the development of theorems parameterized by abstract data and assumptions. The locales automate the process of instantiating the theorems: The user provides concrete instances for the data and discharges the assumptions; in exchange, they obtain an unconditional version of the theorem for the given instance. For example, for Theorem 4.1 the data parameters are families of transition systems, synchronization predicates between them for modeling the communication, and security properties (views, bounds, and triggers). The proof obligations for instantiating the locale are the assumptions of the theorem: communication-strength, symmetry, and a unique secret issuer.

The compositionality framework took 5700 LOC and was built on top of a previously formalized framework for BD security (consisting of 1800 LOC). The system specification comprises 1500 LOC. As expected, the verification of the concrete instances (listed in Section 4.4.3) constituted the bulk of the development, 14400 LOC. The verification of each instance had two components: (1) proofs for the security of individual nodes and (2) verifications of the conditions for compositionality (leading to the corresponding instantiation of the locale).

For all but one case (remote friendship), we started with properties of CoSMed and split them into secret issuer and receiver properties for CoSMeDis, as prescribed by our heuristic. The original proofs for CoSMed were elaborate interactive proofs by unwinding [BPPR16, Section 4]. Their adaptation to CoSMeDis, with strengthening the attacker power, also went according to our heuristic. Nevertheless, this was laborious: The original proof scripts broke in places located deep inside nested case distinctions, hence to adapt them we needed to analyze large proof contexts. Due to the more complex bounds, the proofs for secret issuers were larger (and more time consuming) than those for the receivers. The average size of a proof for the former was 1500 LOC, about twice as large as for the latter.

In contrast to the unwinding proofs, the verification of the compositionality conditions was almost entirely automatic—thanks to the conditions being local (involving only actions and states, no traces or sequences of secrets). Indeed, a case distinction

on all the CoSMeDis actions followed by "auto" (Isabelle's main automatic proof method) was usually sufficient, after instrumenting "auto" with the necessary simplification rules. The average size of an instantiation file, 150 LOC, mainly reflects the boilerplate for locale instantiation.

Overall, the verification of CoSMeDis consists of 21600 LOC, which required 4 person-months (including the formalization of the abstract framework, which involved trial and error). These followed as an extension of CoSMed's verification, consisting of 10000 LOC and having required 3 person-months. A large part of the CoSMeDis proofs were not developed from scratch, but as adaptations of previous CoSMed proofs. Together, these adapted proofs form 10000 LOC, of which roughly 8000 LOC are reused from CoSMed.

4.5. Future Work

Our current implementation of CoSMeDis is a prototype, and there are some features that would need to be added before it can be used in production. In particular, the system currently runs purely in memory (except for the post image files). Hooking it up to a database or another data persistence facility would require some adaptation of the code generator setup and some thought about the information flows to and from the database. It would be interesting future work to study whether a compositional approach could be used to combine security guarantees of the web application and a database backend.

Our prototype has only a minimal set of features. For example, it does not currently support searching for posts whose titles match a string, re-posting a friend's post, offering lists of potential friends based on one's profile, or tagging of posts by its readers. Some of these features, e.g., searching, do not open new channels, hence can be added without affecting the security guarantees. Other features would require amending the security properties. For example, if we wanted to allow re-posting of private posts, this would require acknowledging a new legitimate flow in the bound. Care must also be taken if we want to keep using the compositionality result with simplifying assumptions for the extended system. For example, re-posting of remote posts would need to be implemented in a way that preserves the assumption of a unique secret issuer. This can be achieved by notifying the original node and letting it further propagate the re-post in the network, in line with the single-master paradigm for secrets.

Another interesting direction of future work is studying alternative architecture choices for a system like CoSMeDis. There are distributed architectures for social media platforms based on peer-to-peer (P2P) networks, avoiding the client-server paradigm [BSVD09, CMS09, JNM+12]. They place the application logic on the clients instead of servers, and rely on data encryption to protect the content of communication from servers and their operators. This has the advantage that users do not have to trust these operators or the implementations of the servers. However, despite encryption and decentralization, we believe that security goals and challenges

similar to those of our case study also arise when using such architectures—after all, confidential information is processed in plain text by the client software running on the end-points of the network. For example, a high-level security guarantee about the confidentiality of private posts would require each client in such a P2P network to preserve the confidentiality of posts received from the clients of friends. In principle, such a requirement could again be formulated as a BD Security property of an I/O automaton modeling the behavior of the client. However, the more complex communication topology goes beyond the single-master paradigm that we followed for CoSMeDis: In a P2P network, data is typically forwarded and stored along dynamic paths through the network, which collides with our assumption of a unique secret issuer. In order to compose security properties of the nodes in such a network, we would have to instantiate the more general compositionality result of Chapter 3. In Chapter 5 we will discuss a scenario where we fully specify the scheduling dependencies of secret and observable events in the bound, allowing us to drop the unique secret issuer assumption.

4.6. Related Work

CoSMeDis belongs to a small, but expanding club of running systems proved to be secure using proof assistants, which includes an aircraft microprocessor [HSY06] (in ACL2), a hardware architecture with information flow primitives [dACD+14] (in Coq), a separation kernel [DGK+13] (in HOL4), a noninterferent operating system kernel [MMB+13] (in Isabelle/HOL), a secure browser [JTL12] (in Coq), and an e-voting system [KTB+15] (using the KeY theorem prover jointly with the Joana information flow analyzer).

In practice, security requirements in web applications are typically implemented using *access control*. In particular, for online social networks, relationship-based access control [FAZ09, PS14] supports policies depending on connections in the social graph, e.g., friendship links. Here, we aimed for more than access control: We wanted to protect secrets not only from direct illegitimate access, but also from leaking to unintended recipients who draw inferences based on observations of the system. Such inferences can easily evade access control. Hence, we aim for *information flow control*.

As outlined in Section 3.5, research in information flow security has a rich history. The compositionality result of [BPPR17] discussed in Section 4.3 is closely related to the result presented in Chapter 3, in that the latter incorporates and generalizes the former. The result of Chapter 3 is also inspired by the MAKS framework [Man02]. In that context, the assumptions of observable network traffic and strong and compatible communication infrastructures discussed in Section 4.3 fall into the first case of Mantel's Generalized Zipping Lemma [Man02, Lemma 1], since they imply that there are no neutral communication events. The MAKS framework does not require a unique secret issuer: Its security properties are fine-tuned so that matching problems due to the scheduling of secrets cannot occur. Compared

to both MAKS and Theorem 3.4, the applicability of Theorem 4.1 is restricted by the simplifying assumptions we made in this chapter. However, we believe that our case study demonstrates that there are interesting and realistic systems where these assumptions hold. This suggests that the compositionality result discussed in this chapter provides a useful combination of expressivity (allowing us to compose very fine-grained security policies) and compositionality (requiring only specific assumptions that are easy to check).

4.7. Conclusion

In this chapter, we described a case study of compositional verification of a realistic system, conducted in collaboration with the co-authors of [BPPR17]. In summary, we have

- presented a prototype of a distributed social media platform with formally verified confidentiality guarantees, called CoSMeDis (Section 4.1);
- discussed how to model and verify the complex confidentiality requirements of CoSMeDis (Section 4.2), including dynamic declassification triggers that may be repeatedly fired and released (Section 4.4.3);
- discussed a compositionality result that leverages a set of simplifying assumptions, thereby avoiding the more complex side conditions of Chapter 3, as well as supporting theorems for combining multiple independent secret sources and translating BD Security properties (Section 4.3);
- discussed the application of these results to CoSMeDis (Section 4.4), lifting the security guarantees of a single CoSMed system all the way to an arbitrary network of CoSMeDis nodes;
- reported on the formalization of the above results in the interactive theorem prover Isabelle/HOL. The source code of the formalization is available from the CoSMeDis website.[8]

The compositionality result developed for this case study is formulated in general terms and can potentially be applied to other systems. We have summarized the main steps for instantiating it in Section 4.4.1.

The security properties we proved cover the application logic in the function core of the system, but we have not verified other parts of the system (cf. Section 4.5). In particular, the manually written code of the layers around the core are currently trusted. Extending the verification scope by combining our results with language-based verification techniques for the outer layers would be an interesting direction for future work.

[8] `http://andreipopescu.uk/CoSMeDis.html`

Chapter 5.

Modular Development of Secure Workflow Systems

5.1. Introduction

In the previous chapter, we derived a security property of a distributed system that allows the run-time deployment of an unbounded number of nodes with identical behavior. In this chapter, we focus on the decomposition of a system specification at development time into heterogeneous components, each responsible for different aspects of the overall system behavior. We build upon previous work on compositional verification using the MAKS framework [HMSS07, BH14a]. We focus on workflow systems, because

- they often have interesting security requirements, since they involve multiple participants and sensitive data, and
- they have a natural decomposition into *activities*.

Hence, we can view the workflow as a composition of communicating activities. Human participation is modeled as (local) interaction events between an activity and the users involved. Once an activity is finished, it sends a trigger to its successor activities in the workflow, signaling them to start, along with any data they need as input from the current activity. This means that the activities do not share state, but pass messages: the workflow is enacted by an orchestration of activities communicating data and control flow information.

The advantage is that we can break down the verification of the workflow into verification tasks of the individual activities. This is useful especially if some or all of the activities are simple enough to be trivially secure or amenable to automatic verification, e.g. using language-based techniques or model checking.

Moreover, this architecture enables a stepwise development process. Starting with a very abstract specification of the workflow that is built using generic tasks that are trivially secure or can be verified automatically, we can iteratively refine those tasks to sub-workflows with more detailed security requirements. The sub-workflow is connected to the parent workflow via composition and a suitable routing of data and control flow messages. This expansion of an abstract activity into a more concrete sub-workflow is a kind of *architectural refinement*.

Our running example is a variant of the hiring workflow of Chapter 2, specified using stepwise refinement of a highly abstract initial workflow. We consider both the confidentiality of data (such as the medical report) and activities (such as the occurrence of sensitive diagnostic activities, like drug screenings). For this purpose, we consider those events as secret that represent the input of confidential data as well as the beginning and end of secret activities. As usual, we verify that there is no unintended interdependency between observable and secret behavior. *Intended* dependencies, i.e. allowed information flows, are specified in the declassification bound (cf. Section 2.2.5, p. 17). The declassification bound of the overall workflow results from a combination of the declassification bounds of the individual activities and subprocesses. For this purpose, we have to choose security views and bounds for these components that compose to the desired global security property. For example, note that fully secret activities trivially satisfy a BD Security property, because they do not have observable behavior. However, we have to verify that the observable behavior of the *other* parts of the workflow does not depend on the occurrence of the secret activity.

We formalize this by lifting the MAKS-based approach of [BH14a] to the compositionality results for BD Security presented in Chapter 3. Note that we cannot use the results from Chapter 4, because the simplifying assumptions of that result do not hold here: Neither is the production of secrets limited to only one component, nor is every communication observable (in particular, the control flow triggers to and from sensitive diagnostic activities are secret and non-observable). Hence, we have to use the compositionality result for BD Security in its more general form.

In summary, the basic idea discussed in this chapter is to leverage modular system specifications in order to perform localized refinements. Assuming the system is specified in terms of several components, e.g. implementing different parts of a workflow we can refine each of those components individually, possibly with multiple nested refinement steps, while keeping the other parts of the system fixed. The challenges for compositionally obtaining *security* guarantees in this setting are, as usual,

- choosing local security properties for the components that are strong enough for composition, and
- combining those properties, in particular the declassification bounds, to a security property of the overall system.

We illustrate this approach using the compositional workflow specifications of [BH14a], where a workflow is modeled as a composition of its individual activities. Our running example in this chapter is the familiar hiring workflow. We start with a highly abstract specification and perform two nested refinements of the medical examination.

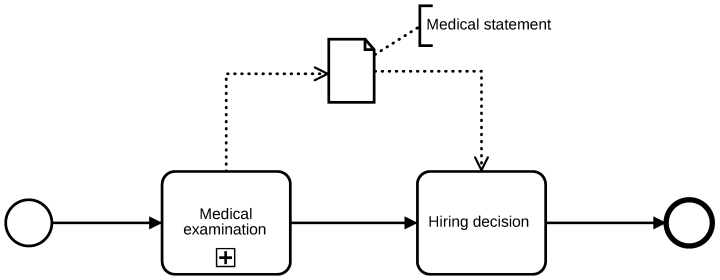

Figure 5.1.: Abstract hiring workflow W_1

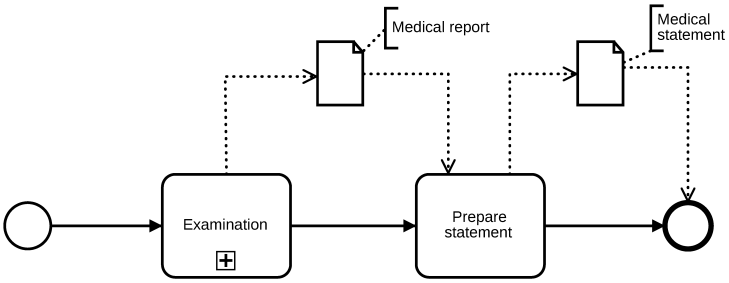

Figure 5.2.: Abstract medical examination workflow W_2

5.2. Example Scenario

Recall the hiring workflow of Chapter 1. In a first step, let us model it using two activities: "Medical examination" and "Hiring decision", depicted in Figure 5.1. The examination produces a medical statement, which is received as input data by the hiring decision activity. The latter is performed by persons in the Human Resources (HR) department, whom we consider to be the observers in this example. In this setting and at this abstraction level, the workflow is trivially secure, because there is no secret information: The medical statement does not contain any confidential details, but only a yes/no summary, which the HR personnel is *allowed*, and in fact required, to receive.

In a second modeling step, we expand the medical activity into a slightly more detailed workflow, depicted in Figure 5.2: "Examination" followed by "Prepare statement". The examination produces a medical report, received by the statement preparation activity. The latter then produces the statement which is to be received by the hiring decision activity of the parent workflow. The details of the medical report are confidential, so we have to verify that the "Prepare statement" activity does not leak more information about the medical details than the binary decision about the candidate's fitness for the job. The security of the overall workflow follows via compositionality.

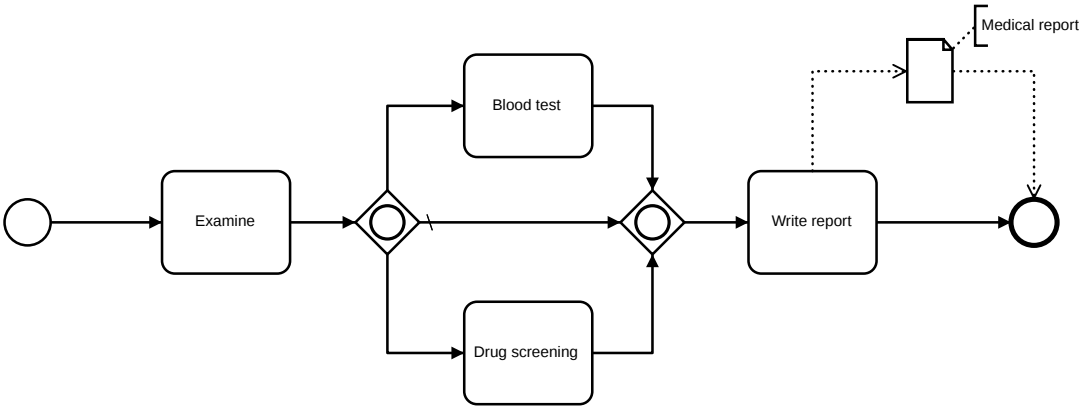

Figure 5.3.: Medical examination subprocess W_3

In a third modeling step, we refine the examination activity into a workflow, depicted in Figure 5.3, with a first medical assessment and eventually the writing of the medical report. In between, there may be further diagnostic activities such as blood tests, depending on whether the result of the first assessment indicates the need for such tests. Hence, the fact *whether* a certain diagnostic activity was performed may already leak information about the condition of the applicant. We therefore classify the sensitive diagnostic activities as completely secret. We now have to verify that it is always possible to insert or remove a diagnostic activity from a trace of this workflow, changing the report, but not the final statement. The declassification bound for the composed workflow is a composition of the bound of this sub-workflow and the bound of the "Prepare statement" activity, synchronized on the content of the medical report. The overall bound still states that only the medical statement is declassified, while the details of the report are kept confidential, and additionally the occurrence of sensitive diagnostic activities.

5.3. System Model

In order to reason about the security of workflows, we need a formal model of workflow specifications and their behavior. We now recall our notion of workflows from [BH14a]. We model workflows as an orchestration of *activities* from a set $\mathcal{A}$, and model the transfer of data and control flow triggers as communication between those activities. For simplicity, we omit aspects such as exceptions or compensation handling, but our definition suffices for our purpose of discussing the verification of security requirements for workflows. We use the special identifier *Env* to denote the *environment* of the workflow, from which the initial input data and trigger is received, and to which the final output data and termination signal is sent. We write $\mathcal{A}_{Env}$ to abbreviate $\mathcal{A} \cup \{Env\}$.

Definition 5.1. A workflow $W = (\mathcal{A}, Docs, CF, DF, U)$ consists of

- a finite set $\mathcal{A}$ of activities,
- a finite set $Docs$ of data items,
- a set $CF \subseteq (\mathcal{A}_{Env} \times \mathcal{A}_{Env})$ of (potential) control flows, where $(a_1, a_2) \in CF$ represents the fact that upon completion of a_1, it may send a trigger to a_2 signaling it to start execution, and
- a set $DF \subseteq (\mathcal{A}_{Env} \times Docs \times \mathcal{A}_{Env})$ of data flows, where $(a_1, d, a_2) \in DF$ represents data item d being an output of activity a_1 and an input to a_2, and
- a finite set U of users participating in the workflow.

The sets $\mathcal{A}$ and $Docs$ correspond to the nodes of a workflow diagram such as Figure 5.3, while CF and DF correspond to the solid and dashed edges, respectively.

We define the behavior of workflows, not in a monolithic way, but in terms of the behaviors of components representing activities communicating with each other. As we discuss in Section 5.5, this simplifies the verification, because it allows us to use a decomposition methodology inspired by [HMSS07] to verify the security of the overall system by verifying security properties of the components. As outlined above, we believe that such a decomposition approach helps not only in scaling up verification of information flow properties to larger systems, but also in enabling localized refinement of activities into subprocesses.

We model each activity a as a transition system

$$LTS_a = (St_a, \sigma_a^0, Ev_a, In_a, Out_a, \rightarrow_a)$$

analogously to the individual components of multi-agent systems defined in [HMSS07, Definition 3]. It has events of the form

1. $Start_a(u)$, starting the activity a and assigning it to the user $u \in U$,
2. $Setval_a(u, i, v)$ and $Outval_a(u, i, v)$, representing a user $u \in U$ reading (or setting, respectively) the value v of data item i,
3. $End_a(u)$, marking the end of the activity,
4. $Recv_a(a', m)$, representing the receiving of a message m from activity $a' \in \mathcal{A}_{Env}$,
5. $Send_a(a', m)$, representing the sending of a message m to activity $a' \in \mathcal{A}_{Env}$, and

Moreover, we allow internal events, used to model, for example, intermediate computations. We consider messages that have the form

- $Trigger$, used to trigger a control flow to a successor activity in the workflow,
- $Data(i, v)$, used to transfer the value v for data item i, and

- $AckData(i)$, used to acknowledge the receipt of a data item.

In particular, $Recv_a(Env, Trigger)$ and $Send_a(Env, Trigger)$ model the receipt or sending of the initial and final control flow triggers from or to the environment, respectively, signaling the beginning and termination of the workflow. Similarly, $Recv_a(Env, Data(i, v))$ and $Send_a(Env, Data(i, v))$ are used to model the exchange of input and output data items with the environment.

The following definition puts this together and classifies events into inputs and outputs.

Definition 5.2. A *specification of activity a in the workflow W* is a transition system

$$LTS_a = (St_a, \sigma_a^0, Ev_a, In_a, Out_a, \rightarrow_a)$$

where

$$\begin{aligned}
Msg &= \{Trigger\} \cup \{Data(i, v) \mid i \in Docs, v \in V\} \cup \{AckData(i, v) \mid i \in Docs\} \\
In_a &= \{Start_a(u) \mid u \in U\} \cup \{Recv_a(a', m) \mid a' \in \mathcal{A}_{Env}, m \in Msg\} \cup \\
&\quad \{End_a(u) \mid u \in U\} \cup \{Setval_a(u, i, v) \mid u \in U, i \in Docs, v \in V\} \\
Out_a &= \{Send_a(a', m) \mid a' \in \mathcal{A}_{Env}, m \in Msg\} \cup \\
&\quad \{Outval_a(u, i, v) \mid u \in U, i \in Docs, v \in V\} \\
Ev_a &\supseteq In_a \cup Out_a
\end{aligned}$$

Using separate messages for data and control flows is inspired by the BPMN standard, which describes its (informal) execution semantics in terms of tokens that are passed from one activity to the next, representing control flow separately from data flows. Indeed, the data items produced as output by one activity may not all be consumed by the immediate successor activity, but by different activities further down in the workflow. Moreover, splitting the output of an activity into different events for each data item and control flow trigger simplifies the modeling of confidentiality, as it becomes straightforward to classify events transporting *Data* messages into confidential or non-confidential events based on the classification of the data items they transport.

We denote a few specific sets of events as follows:

- $Ev_W = \bigcup_{a \in \mathcal{A}} Ev_a$ is the set of all events in the workflow,
- Ev_u for $u \in U$ is the set of events involving the user u, i.e. the events of the categories 1–3 above for the given user u,
- $Ev_U = \bigcup_{u \in U} Ev_u$ is the set of all user interaction events,
- Ev_i for $i \in Docs$ is the set of events involving the data item i, i.e. events of the forms
  - $Setval_a(u, i, v)$ and $Outval_a(u, i, v)$ for some u and v, or

  - $Send_a(b, msg)$ and $Recv_a(b, msg)$ for some a and b and a message msg of the form $Data(i, v)$ or $AckData(i)$ for some v,

- $Ev_{Docs} = \bigcup_{i \in Docs} Ev_i$ is the set of all events involving data items, and
- $Ev_{a,b}$ for $a, b \in \mathcal{A}_{Env}$ is the set of communication events of activity a with b, i.e. events of the form $Send_a(b, msg)$ and $Recv_a(b, msg)$. In particular, $Ev_{a,Env}$ contains the communication events of an activity a with the environment.

Note that Definition 5.2 does not make assumptions about the behavior of the activity, since we do not need any to instantiate the compositionality result. However, any security-relevant aspects of communication patterns that influence or depend on confidential information will have to be encoded in the declassification bounds. From a functional perspective, the activities in a workflow will have to agree on some basic protocols of communication. We formally specify transition relations capturing such a protocol in Appendix A. After initialization, each of our activities waits for messages from other activities, transferring input data or triggering a control flow. When all incoming control flows have been triggered as required by the workflow specification, the activity starts executing. Upon completion, it sends its output data items via the outgoing data associations, and triggers outgoing control flows. In addition to this generic communication behavior, we provide example specifications for a few types of activities in Appendix A. In particular, we define the behavior of

- user activities that allow users to read and write data items, and
- gateways that make a decision on the control flow based on the contents of their input data items.

We use the generic activities of Appendix A as building blocks for our workflow specifications. For example, in Figure 5.1, we model both the medical examination subprocess as well as the hiring decision as a user activity, generating and consuming the medical statement. This is of course an over-simplification, but as outlined above, we refine the medical examination in subsequent modeling steps. For the second step depicted in Figure 5.2, we use another user activity that generates the medical report, and an activity that extracts the statement from it. Figure 5.3 shows a more complex control flow. We use a gateway that triggers the additional specific examinations only if they are indicated by the result of the initial assessment. This can be modeled, for example, by a checkbox in the assessment report. Conversely, the right gateway in Figure 5.2 merges the control flow again by waiting for the completion of those additional examination activities that have been triggered before.

The overall workflow emerges from the composition of these activities: Assuming that they follow the communication protocol, receiving and sending data and triggers according to the workflow specification, together they enact the workflow. We allow the activities to communicate asynchronously; this helps avoid synchronization

problems with activities running at different times during the workflow. Nevertheless, the activities have to synchronize to a certain degree: we need to ensure that an activity has received all its input data items before it is triggered. For this purpose, activities send *AckData* messages to acknowledge the receipt of data items.[1]

We implement asynchronous communication by composing the activities with a communication platform P_W that is responsible for buffering messages. For a specification of the platform, see [HMSS07, Section 2.3]. The notion of composition that we use in this chapter is the one that is used in the MAKS framework [Man02]: components operate in parallel, but synchronize on shared events.

Definition 5.3. Let LTS_1 and LTS_2 with $LTS_i = (St_i, \sigma_i, Ev_i, In_i, Out_i, \rightarrow_i)$ be two labeled transition systems. We say LTS_1 and LTS_2 are *composable* iff

$$(Ev_1 \cap Ev_2) \subseteq (In_1 \cap Out_2) \cup (In_2 \cap Out_1)$$

The *synchronized parallel composition of* LTS_1 *and* LTS_2 is the system

$$LTS = (St, \sigma, Ev, In, Out, \rightarrow)$$

where

$$\begin{aligned} St &= St_1 \times St_2 \\ \sigma &= (\sigma_1, \sigma_2) \\ Ev &= Ev_1 \cup Ev_2 \\ In &= (In_1 \setminus Out_2) \cup (In_2 \setminus Out_1) \\ Out &= (Out_1 \setminus In_2) \cup (Out_2 \setminus In_1) \end{aligned}$$

and the transition relation $\rightarrow$ is defined by the rules in Figure 5.4. We denote this composition as $LTS_1 \parallel LTS_2$ in this chapter.

$$\textsc{Local}_1 \ \frac{\sigma_1 \xrightarrow{e}_1 \sigma_1' \qquad e \in Ev_1 \setminus Ev_2}{(\sigma_1, \sigma_2) \xrightarrow{e} (\sigma_1', \sigma_2)} \qquad \textsc{Local}_2 \ \frac{\sigma_2 \xrightarrow{e}_2 \sigma_2' \qquad e \in Ev_2 \setminus Ev_1}{(\sigma_1, \sigma_2) \xrightarrow{e} (\sigma_1, \sigma_2')}$$

$$\textsc{Comp} \ \frac{\sigma_1 \xrightarrow{e}_1 \sigma_1' \qquad \sigma_2 \xrightarrow{e}_2 \sigma_2' \qquad e \in Ev_1 \cap Ev_2}{(\sigma_1, \sigma_2) \xrightarrow{e} (\sigma_1', \sigma_2')}$$

Figure 5.4.: Transition relation for synchronized parallel composition

The possible behaviors of such a composition corresponds are interleavings of possible traces of the components, synchronized on shared events:

$$[\![LTS_1 \parallel LTS_2]\!] = \{t \in Ev^* \mid (t \restriction Ev_1) \in [\![LTS_1]\!] \wedge (t \restriction Ev_2) \in [\![LTS_2]\!]\}$$

[1] In Chapter 6, we will discuss a refinement of the communication platform so that it guarantees causal delivery. This eliminates the need for acknowledgment messages, but leads to additional proof obligations for compositionality.

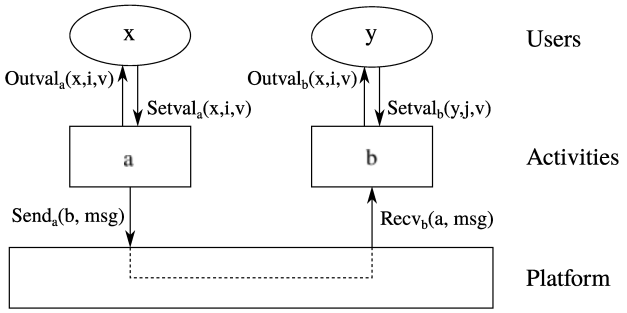

Figure 5.5.: Communication between users, activities and platform

This composition operator corresponds to a special case of the more general notion of composition that we introduced in Chapter 2. We can formulate the former in terms of the latter by choosing the *identity* on the communication events $(Ev_1 \cap Ev_2)$ as the synchronization relation. The canonical composed events of the form $(1, e)$, $(2, e)$, and (e, e') are isomorphic to $Ev_1 \cup Ev_2$ in this case, allowing us to switch between those representations without affecting security. Moreover, for pairwise composable systems, the above composition operator is associative in the sense that $LTS_1 \parallel (LTS_2 \parallel LTS_3)$ and $(LTS_1 \parallel LTS_2) \parallel LTS_3$ have exactly the same interfaces, sets of events, and traces. This makes it straightforward to lift the binary composition operator to n-ary composition.

We use this n-ary composition to formally model a workflow system LTS_W as the composition of the set of components $\mathcal{A} \cup \{P_W\}$, including the activities $\mathcal{A}$ and the platform P_W:

$$LTS_W = (\|_{a \in \mathcal{A}} LTS_a) \,\|\, LTS_{P_W}$$

Intuitively, each activity a synchronizes with the platform on $Send_a(b, m)$ and $Recv_a(b, m)$ events, and the platform buffers messages it receives in a *Send* event and delivers it to the recipient activity using a *Recv* event.

The overall architecture is depicted in Figure 5.5. Upon composition with the platform, the communication events between the activities become internal events of the composed system. Only the communication events with users remain input and output events. These events form the user interface of the workflow system.

5.4. Security Policies

In this chapter, we consider confidentiality requirements regarding the data items processed in workflows. For this purpose, we assign data items to security domains, drawn from a set $\mathcal{D}$. Moreover, we define a flow policy, i.e. a reflexive and transitive relation on domains that specifies from which domains to which other domains information may flow [MSZ06]. Note that, even though we focus on confidentiality in this paper, integrity requirements can be seen as a dual to confidentiality and han-

dled using information flow control. For example, in [MSZ06] a lattice of combined security levels is built as a product of a confidentiality lattice and an integrity lattice. For our example workflow, we only require two confidentiality domains *HR* and *Med*. The medical statement in Figures 5.1 and 5.2 is assigned to the *HR* domain, while the medical report in Figures 5.2 and 5.3 is assigned to the *Med* domain. The example flow policy states that information may flow from *HR* to *Med*, but not vice versa, i.e. $HR \rightsquigarrow Med$ and $Med \not\rightsquigarrow HR$.

Users read and write the contents of data items via the inputs and outputs of activities they participate in. In order to exclude unwanted direct information flows, we have to make sure that the classifications of the data items that users work with are compatible with their clearances. A straightforward approach is to enforce a Bell-LaPadula style mandatory access control. This can be formulated in terms of clearances that are assigned to users, and classifications that are assigned to activities based on the classifications of their inputs and outputs. Users may only participate in an activity if they have a clearance matching the activity's classification.[2] Moreover, the *occurrence* of certain activities might also be considered as confidential, so we additionally classify the information whether an activity has occurred or not.

Definition 5.4. A *policy on the workflow W w.r.t. the domains $\mathcal{D}$* is a tuple $\mathcal{P} = (dom, cl, ocl, ucl, \rightsquigarrow)$ consisting of

- a *flow relation* $\rightsquigarrow \subseteq \mathcal{D} \times \mathcal{D}$,
- a *data classification* function $dom\colon Docs \to \mathcal{D}$,
- an *activity classification* function $cl\colon \mathcal{A}_{Env} \to \mathcal{D}$,
- an *activity occurrence classification* function $ocl\colon \mathcal{A}_{Env} \to \mathcal{D}$, and
- a *user clearance* function $ucl\colon U \to \mathcal{D}$.

For each security domain $d \in \mathcal{D}$, the data classification partitions the data items of a workflow into "high" data items $Docs_d^H$, which are confidential in domain d, and "low" data items $Docs_d^L$, from which the policy allows information to flow to d.

$$Docs_d^H = \{i \in Docs \mid dom(i) \not\rightsquigarrow d\} \qquad Docs_d^L = \{i \in Docs \mid dom(i) \rightsquigarrow d\}$$

Similarly, we denote the sets of *activities* that are classified lower or higher than a given domain d, and the sets of *users* with a clearance lower or higher than d, as follows:

$$\mathcal{A}_d^H = \{a \in \mathcal{A}_{Env} \mid cl(a) \not\rightsquigarrow d\} \qquad \mathcal{A}_d^L = \{a \in \mathcal{A}_{Env} \mid cl(a) \rightsquigarrow d\}$$
$$U_d^H = \{u \in U \mid ucl(u) \not\rightsquigarrow d\} \qquad U_d^L = \{u \in U \mid ucl(u) \rightsquigarrow d\}$$

Finally, the set of activities whose *occurrence* is confidential for d, which we call "secret" activities, is defined as

$$\mathcal{A}_d^S = \{a \in \mathcal{A}_{Env} \mid ocl(a) \not\rightsquigarrow d\}$$

[2] This access control is specified in Appendix A, together with the behavior of the activities in our example workflows.

Table 5.1.: Classifications of example activities

Workflow	Activity	*cl*	*ocl*
W_1	Medical examination	*Med*	*HR*
W_1	Hiring decision	*HR*	*HR*
W_1	*Env*	*HR*	*HR*
W_2	Examination	*Med*	*HR*
W_2	Prepare statement	*Med*	*HR*
W_2	*Env*	*HR*	*HR*
W_3	Examine	*Med*	*HR*
W_3	Blood test	*Med*	*Med*
W_3	Drug screening	*Med*	*Med*
W_3	Write report	*Med*	*HR*
W_3	*Env*	*Med*	*HR*

Table 5.2.: Classifications of example data items

Data item	*dom*
Medical report	*Med*
Medical statement	*HR*

In our example workflows, only the specific examination activities in W_3 (cf. Figure 5.3) have to be kept completely secret from the HR department. We therefore classify their occurrence as *Med*. The occurrence of any of the other activities is not confidential and is classified as *HR*. All activities except for the "Hiring decision" activity in W_1 (cf. Figure 5.1) process confidential data. Hence, their *content* is classified as *Med*. Moreover, the *environment* of W_3 is classified as *Med*, because this workflow emits confidential output to its environment. We summarize the classifications of activities and data items in Tables 5.1 and 5.2, respectively.

Our goal is to verify that the workflow system does not leak more information than intended to a user in a domain d about

- the content of data items i that are confidential for d, and
- the occurrence of activities a that are secret for d.

As usual, we formulate a BD Security property that captures these requirements. We first define a suitable view on the workflow system.

Definition 5.5. Let $\mathcal{P}$ be a policy on the workflow W w.r.t. $\mathcal{D}$. The *view on* W

for $d \in \mathcal{D}$ *in* $\mathcal{P}$ is defined as $\mathcal{V}_d = \big(Ev_d^{obs}, \mathrm{id}, Ev_d^{sec}, \mathrm{id}\big)$, where

$$\begin{aligned} Ev_d^{obs} &= \bigcup_{u \in U_d^L} Ev_u \cup \bigcup_{i \in Docs_d^L, a \in \mathcal{A}} (Ev_i \cap Ev_{a,Env}) \\ Ev_d^{sec} &= \big\{ Setval_a(u, i, val) \mid i \in Docs_d^H \big\} \\ &\quad \cup \big\{ Send_a(b, Trigger) \mid a \in \mathcal{A}_d^S \vee b \in \mathcal{A}_d^S \big\} \end{aligned}$$

The set of observable events consists of all events performed by any user in the domain d or lower. Moreover, we consider non-confidential inputs and outputs of the workflow to be observable, i.e. communication events with the environment transporting data items in the domain d or lower. For example, in the workflow W_2 (cf. Figure 5.2), the output event sending the medical statement to the environment is observable for domain *HR*.

For the secret events, we choose two subsets representing the two categories of confidential information outlined above: input events changing the content of confidential data items, and events sending execution triggers to or from secret activities. Note that we only consider *input* events of data items as confidential here, and not output or processing events. This is adequate in scenarios where the ultimate *source* of confidential information is always user input (which is the case in our hiring workflow example). If the system itself *generates* confidential information internally, then the events representing that generation would have to be added to Ev^{sec}. For the workflow W_2, the set of secret events for *HR* consists of the *Setval* events for the medical report.

The next step is to specify a declassification bound, since it is typically neither possible nor desirable to verify the absence of *any* information flow from secret to observable events. Indeed, recall that in the hiring workflow, in particular in the subprocess depicted in Figure 5.2, the equivalence class of the confidential medical report is released into the observable medical statement. Our original work in [BH14a] was based on the MAKS framework [Man00a], which does not support control over *what* information is declassified. In [BH14a], we used a workaround that only allows control over *who* may declassify information: we introduced the notion of *trusted users*, and moved the responsibility for declassifying information out of the system and into the hands of those users. The scope of the verification was therefore limited to ensuring that the system does not leak additional confidential information.

BD Security allows a more fine-grained control over what is declassified. For example, let us specify that the workflow W_2 (cf. Figure 5.2) does not leak more about the content of the medical report than its equivalence class. With respect to the view for the *HR* domain as defined above, the bound

$$B = \{(sl, sl') \mid sl \sim_{Med} sl'\}$$

formalizes this,[3] where $\sim_{Med}$ denotes the lifting of the equivalence class $\approx_{Med}$ on

[3] For simplicity, this bound also allows the *number* of updates to the report to be declassified; for a variant of the bound that forbids this as well, see Section 2.2.5.

medical reports to workflow events, i.e.

$$Setval_a(u, Report, x) \sim_{Med} Setval_a(u, Report, x') \longleftrightarrow x \approx_{Med} x'$$
$$Send_a(b, Data(Report, x)) \sim_{Med} Send_a(b, Data(Report, x')) \longleftrightarrow x \approx_{Med} x'$$
$$Recv_a(b, Data(Report, x)) \sim_{Med} Recv_a(b, Data(Report, x')) \longleftrightarrow x \approx_{Med} x'$$

and $\sim_{Med}$ is the identity on events that do not contain medical reports. Note that, in $\mathcal{V}_{HR}$ as defined above, events of the form $Setval_a(u, Report, x)$ are the only confidential events in W_2, because only input of data items is considered confidential (not their communication from one activity to another), the medical report is the only confidential data item, and there are no secret activities.

5.5. Compositional Verification

We now describe how to compositionally verify that a given workflow system satisfies BD Security with respect to such views and bounds. To ease the verification, we make use of the methodology presented in [BH14a] (based on the one originally presented in [HMSS07]) to verify the resulting distributed system, decomposing the workflow into its individual activities. This has several advantages. The verification of individual activities is likely to be easier than the verification of the complete system, due to the smaller size of the components. It also facilitates reuse of verified activities in other workflows. Finally, as we will show below, it enables a stepwise development process of secure workflow systems. The disadvantage, as we have learned in previous chapters, is that we have to find and verify local security properties of the components that are strong enough to be compositional.

For each domain $d \in \mathcal{D}$, we verify that there are no unintended flows of confidential information to users in that domain. With the discussions of Chapter 3 in mind, it is easy to see that the views $\mathcal{V}_d$ defined above are too weak for compositional verification. Indeed, $\mathcal{V}_d$ mostly classifies internal events of the components as secret or observable, while most communication events between the activities are treated as neutral.

Recall that our compositionality result in Chapter 3 requires that, in each pair of matching communication event, *at least one* event, namely the *input* event, is observable or secret. Hence, in order to decompose a global view on a workflow system into local views on the individual activities, we require that only *Send* events may be neutral for an activity, while *Recv* events must be observable or secret. More specifically, a *Send* event is allowed to be neutral iff the corresponding *Recv* event is secret and non-observable. If either one of them is observable, the other has to be observable as well. This guarantees that the local views "fit together" in the sense of Definition 3.7.

Definition 5.6. Let $\mathcal{V}$ be a view on a workflow system LTS_W. A family $(\mathcal{V}_a)_{a \in \mathcal{A}}$ of views $\mathcal{V}_a = (Ev_a^{obs}, \mathrm{id}, Ev_a^{sec}, \mathrm{id})$ is a *valid decomposition of* $\mathcal{V}$ iff, for all $a, b \in \mathcal{A}$,

- $Ev_{\mathcal{V}}^{obs} \cap Ev_a \subseteq Ev_a^{obs}$ and $Ev_{\mathcal{V}}^{sec} \cap Ev_a \subseteq Ev_a^{sec}$

- $Recv_b(a,m) \in Ev_b^{sec} \cup Ev_b^{obs}$
- $Recv_b(a,m) \in Ev_b^{obs} \longleftrightarrow Send_a(b,m) \in Ev_a^{obs}$
- $Send_a(b,m) \in Ev_a^{sec} \longrightarrow Recv_b(a,m) \in Ev_b^{sec}$
- $Recv_b(a,m) \in Ev_b^{obs} \cap Ev_b^{sec} \longrightarrow Send_a(b,m) \in Ev_a^{sec}$

Since we assume for simplicity in this chapter that the observations and secrets are the events themselves (i.e. *getObs* and *getSec* are the identity), the remaining conditions on views dealing with equivalent events in Definitions 3.7, 3.15, and 3.20 are trivially satisfied.

Also one of the well-behavedness conditions of Lemma 3.3 is trivially satisfied, namely the replacement of input events matching view-equivalent outputs. The other conditions have to be satisfied by the components, however, namely the acceptance of neutral inputs and the flexible scheduling of secrets and observations.

Theorem 5.1. *Let $(\mathcal{V}_a)_{a\in\mathcal{A}}$ be a valid decomposition of a view $\mathcal{V}$ on a workflow system LTS_W. Let*

$$N_a = \left\{ Recv_a(b,m) \mid Send_b(a,m) \notin Ev_b^{sec} \cup Ev_b^{obs} \right\}$$

be the set of receiving events of activity a corresponding to neutral sending events. If

1. *LTS_a satisfies BD Security w.r.t. $\mathcal{V}_a$ and B_a for all $a \in \mathcal{A}$,*
2. *either $N_a = \emptyset$, or B_a is total in N_a and LTS_a additionally satisfies eager insertion of N_a (cf. Definition 3.8), insertion and deletion of N_a before $Ev_a^{obs} \setminus Ev_a^{sec}$ without communication events in between (cf. Definitions 3.9 and 3.18), and backwards-strict BD Security (cf. Definition 3.24) w.r.t. $\mathcal{V}_a$ and B_a for all $a \in \mathcal{A}$, and*
3. *one of the following holds:*
   a) *$Ev_a^{obs} \subseteq Ev_a^{sec}$ for all $a \in \mathcal{A}$, or*
   b) *$Ev_a^{sec} \setminus N_a \subseteq Ev_a^{obs}$ for all $a \in \mathcal{A}$, or*
   c) *LTS_a supports eager insertion of $Ev_a^{sec} \setminus Ev_s^{obs}$ as well as insertion and deletion of $Ev_a^{sec} \setminus Ev_s^{obs}$ before $Ev_a^{obs} \setminus Ev_a^{sec}$ without communication events in between w.r.t. $\mathcal{V}_a$ and B_a for all $a \in \mathcal{A}$,*

then LTS_W satisfies BD Security w.r.t. $\mathcal{V}_\mathcal{A} = (Ev_\mathcal{A}^{obs}, \mathrm{id}, Ev_\mathcal{A}^{sec}, \mathrm{id})$ and $B_\mathcal{A}$ with

$$Ev_\mathcal{A}^{obs} = \bigcup_{a\in\mathcal{A}} Ev_a^{obs}$$

$$Ev_\mathcal{A}^{sec} = \bigcup_{a\in\mathcal{A}} Ev_a^{sec} \setminus N_a$$

$$B_\mathcal{A} = \big\{ (sl, sl') \mid \forall a \in \mathcal{A}.(sl \restriction Ev_a^{sec}, sl' \restriction Ev_a^{sec}) \in B_a \wedge sl \equiv_{Ev_\mathcal{A}^{obs}} sl' \wedge causal(sl') \big\}$$

where $causal(sl')$ *specifies that the communication events in* sl' *occur in a causal order, i.e. each message is received only* after *it has been sent, and each sent message is received at most once, defined as follows, where* $sentMsgs(a, b, t)$ *and* $rcvdMsgs(b, a, t)$ *return the multiset of messages sent from* a *to* b *or received by* b *from* a*, respectively, in the trace* t*, and* $\subseteq_\#$ *denotes the multiset ordering:*

$$causal(sl') \equiv \forall t \leq sl'.\, \forall a, b \in \mathcal{A}.\, rcvdMsgs(b, a, t) \subseteq_\# sentMsgs(a, b, t)$$

This theorem allows us to compose the security guarantees of individual activities to a security guarantee of the overall workflow. The views and declassification bounds of the activities are combined as usual. The only difference to the bound composition of previous chapters is the *causal* predicate, which factors in *asynchronous* communication: the activities do not synchronize with each other directly, but communicate via a platform that buffers messages. The *causal* predicate captures the fact that messages are only received (once) after they have been sent.

The price for compositional verification is that, first, the local views have to be chosen strong enough to satisfy the conditions of Definition 5.6, and second, the activities must be flexible w.r.t. the acceptance of neutral inputs and the scheduling of secrets and observations. As we have seen in the previous chapter, it is possible to avoid having to prove these additional side conditions in certain scenarios. In Chapter 4, we made several simplifying assumptions. In particular, we strengthened the local views so that *all* communication events are observable, to some degree. This means that the conditions regarding neutral events are trivially satisfied. Moreover, we assumed that (local) secrets are only produced by *one* node in the network. This means that the condition regarding the scheduling of secrets is trivially satisfied.

For our workflow systems, these assumptions do not apply: in particular, we cannot treat all communication events as observable, because (completely) secret activities have to be invisible for observers. Nevertheless, we do not need the full generality of Theorem 5.1 for the simple example workflows in this chapter. In particular, we do not need neutral events at the communication interfaces. Hence, we make two simplifying assumptions that are somewhat dual to those of Chapter 4. First, we treat all communication events as potentially *secret*, to some degree. Second, we assume that all observable events produce a secret *as well*, and that the scheduling of observable and (proper) secret events is fully specified in the declassification bound. This might seem counterintuitive, but note that, while the content of observable events is not secret, the occurrence and timing of observable events *can* influence the possible occurrences of secret events. For example, the medical report is secret in the hiring workflow, but the *occurrence* of the medical examination activity is observable, so an observer can deduce that the secret report cannot have been written if the corresponding activity has not been started yet. Fully specifying such dependencies of secrets on observations in the declassification bound leads to more precise security guarantees that are easier to compose. Indeed, the side conditions of Theorem 5.1 are trivially satisfied with these assumptions, due to $N_a = \emptyset$ and $Ev_a^{obs} \subseteq Ev_a^{sec}$.

However, the disadvantage is that we are forced to specify the valid sequences and scheduling of secret events relative to observable events in the declassification bound. In the examples of this chapter, this is a relatively small inconvenience: these additional specifications only refer to static knowledge about the communication patterns of the activities, for example that each activity sends its outgoing data items before it sends execution triggers to its successors.

These considerations lead us to the following local views. We consider all inputs and outputs as observable *except* those that

- belong to or communicate with secret activities,
- transport confidential data items, or
- are performed by high users.

Conversely, we consider *all* input and output events as potentially secret. The precise dependencies between observable and secret inputs and outputs, if any, have to be specified in the declassification bound.

Definition 5.7. Let $d \in \mathcal{D}$ be a domain, and $a \in \mathcal{A}$ be an activity for d. We define the *local view on a for d w.r.t. $\mathcal{P}$* as $\mathcal{V}_a^d = (Ev_a^{obs}, \mathrm{id}, Ev_a^{sec}, \mathrm{id})$ with

$$Ev_a^{obs} = (In_a \cup Out_a) \setminus \left(\bigcup_{a' \in \mathcal{A}_d^S} \left(Ev_{a'} \cup Ev_{a,a'}\right) \cup \bigcup_{i \in Docs_d^H} Ev_i \cup \bigcup_{u \in U_d^H} Ev_u \right)$$

$$Ev_a^{sec} = (In_a \cup Out_a)$$

Combining these local views, we define the *global view on W for d w.r.t. $\mathcal{P}$* as

$$\mathcal{V}_{\mathcal{A}}^d = \left(Ev_{\mathcal{A}}^{obs}, \mathrm{id}, Ev_{\mathcal{A}}^{sec}, \mathrm{id}\right)$$

where

$$Ev_{\mathcal{A}}^{obs} = \bigcup_{a \in \mathcal{A}} Ev_a^{obs} \qquad\qquad Ev_{\mathcal{A}}^{sec} = \bigcup_{a \in \mathcal{A}} Ev_a^{sec}$$

Note that the global view $\mathcal{V}_{\mathcal{A}}^d$ is *stronger* than the original view $\mathcal{V}_d$ we chose in Definition 5.5 in the sense that more events are considered observable or secret in a domain d. This allows us to use the translation Theorem 4.2 in order to derive a security guarantee w.r.t. $\mathcal{V}_d$ *after* we have obtained a security guarantee w.r.t. $\mathcal{V}_{\mathcal{A}}^d$ using the compositionality result of Theorem 5.1.

For example, let us informally discuss how to apply these theorems to the workflow W_2. On this abstraction level, we model the medical examination as a simple user activity that gets the medical report from user input. (This will be refined to a more detailed workflow of its own in the next section.) Such a user activity collects input data items from predecessor activities and from user input, and outputs them to successor activities in the workflow. Beyond observable events, it only declassifies the static knowledge that

- the output is correlated to the input, i.e. the output for a data item is the last version that was input, and
- the output is sent directly before the next activity is triggered.

We capture these correlations in a predicate $static_{Exam}$ and use it in the following bound:

$$B_{Exam} = \left\{ (sl, sl') \mid static_{Exam}(sl') \wedge sl \equiv_{Ev_{\mathcal{A}}^{obs}} sl' \right\}$$

We omit the full definition of this and other bounds and their local verification in this chapter, since we focus on compositional reasoning here rather than on the local verification of activities. For our earlier work in the context of the MAKS framework, we had verified local security properties of activities in Isabelle/HOL using an unwinding proof technique [Man00b]. We refer to Appendix B of the extended version of [BH14a] for details.

The other activity in W_2, "Prepare statement", extracts the equivalence class of the (secret) medical report and produces a corresponding (observable) statement. Hence, the equivalence class is declassified. The static knowledge only consists of the fact that the confidential input of the medical report has to arrive before the activity starts executing.

$$B_{Stmt} = \left\{ (sl, sl') \mid sl \sim_{Med} sl' \wedge static_{Stmt}(sl') \wedge sl \equiv_{Ev_{\mathcal{A}}^{obs}} sl' \right\}$$

By instantiating Theorem 5.1, we obtain a security property of the workflow w.r.t. $\mathcal{V}_{\mathcal{A}}^{d}$ and the bound

$$B_{W_2} = \left\{ (sl, sl') \mid (sl \restriction Ev_{Stmt}) \sim_{Med} (sl' \restriction Ev_{Stmt}) \wedge static_{W_2}(sl') \wedge sl \equiv_{Ev_{\mathcal{A}}^{obs}} sl' \right\}$$

where

$$static_{W_2}(sl') = static_{Exam}(sl' \restriction Ev_{Exam}) \wedge static_{Stmt}(sl' \restriction Ev_{Stmt}) \wedge causal(sl')$$

This connects the bounds of the two activities: the "Prepare statement" activity declassifies the equivalence class of the received report, which (due to the causality of communication) equals the report sent by the "Examination" activity, which equals the final version of the report that was received as input from *Med* users. Effectively, the workflow declassifies the equivalence class of the final report to *HR* users via its observable output, the medical statement.

This is stronger than the bound we formulated in Section 5.4 (w.r.t. the weaker view $\mathcal{V}_d$ of Definition 5.5), namely

$$B = \{ (sl, sl') \mid sl \sim_{Med} sl' \}$$

Using the translation theorem from Chapter 4, we prove that the security of LTS_{W_2} w.r.t. $\mathcal{V}_{\mathcal{A}}^{d}$ and B_{W_2} implies security w.r.t. $\mathcal{V}_d$ and B. We have to show that, for each pair of secrets in the desired bound B and for each system trace t, we can construct a pair of secrets in the more precise bound B_{W_2}, which is easy: copy the observable events from t and add secret communication events between the activities, transporting the final version of the report.

5.6. Workflow Refinement

In the previous section, we discussed an approach to compose the security guarantees of individual activities to one of an overall workflow. For larger workflows, we might not be willing or able to construct the workflow directly in full detail. This holds in particular in the early stages of the development of a workflow, where we do not yet know how some of its activities will be implemented in detail. In such cases, we proceed in a stepwise manner, starting with a rough draft of the workflow and iteratively expanding abstract activities into sub-workflows, as we have sketched for the example scenario in Section 5.2. In this section, we discuss how the verification can be performed iteratively, as well. The idea is to plug the security guarantee of a sub-workflow into that of the parent workflow via compositional reasoning.

5.6.1. Specification of Workflows with Embedded Subprocesses

We begin by formalizing a notion of workflow refinement. Recall the definition of workflow specifications $W = (\mathcal{A}, Docs, CF, DF, U)$ with activities $\mathcal{A}$, documents $Docs$, control flows CF, data flows DF, and users U. In addition to the activities $\mathcal{A}$, we use the constant placeholder name Env for the environment, allowing input and output from or to the outside. We use this placeholder Env as a hook-up point to embed a sub-workflow into a parent workflow: the parent becomes the environment of the sub-workflow. Let W be a workflow where we want to refine an abstract activity $a \in \mathcal{A}_W$ to a subprocess W'. We hook together the two workflows by

- substituting a for Env in the subprocess W', so that its initial activities expect input data and control flow triggers from an activity named a, and its final activities send output data and a control flow trigger back to a,
- replacing the behavior of a in the parent workflow W by a proxy activity that collects input data for the subprocess from other activities in the parent workflow according to its specification, forwards data and trigger to initial activities in W', waits for data and trigger sent by final activities in W' and forwards them to subsequent activities in the parent workflow.

The advantage of this approach that it requires minimal changes in the two workflows; the work of hooking them together is performed by the proxy activity that is substituted for a. The workflow specification of the merged process is defined as follows:

Definition 5.8. Let W and W' be two workflow specifications. W' is a *valid refinement of a in W* iff

- $\mathcal{A}_W \cap \mathcal{A}_{W'} = \emptyset$,
- the inputs that activities in W' expect from the environment coincide with the outputs that activities in the parent workflow W provide to a, i.e., for all $a' \in W'$, $(Env, i, a') \in DF_{W'}$ iff there is some activity a_0 in W with $(a_0, i, a) \in DF_W$, and

- the outputs that activities in W' provide to the environment coincide with the inputs that activities in W expect to receive from a, i.e., for all $a' \in W'$, $(a', i, Env) \in DF_{W'}$ iff there is some activity a_0 in W with $(a, i, a_0) \in DF_W$.

The *embedding of W' into W in place of a*, denoted as $W[a \leftarrow W']$, is a workflow specification $(\mathcal{A}, Docs, CF, DF, U)$ with

$$\begin{aligned} \mathcal{A} &= \mathcal{A}_W \cup \mathcal{A}_{W'} \\ Docs &= Docs_W \cup Docs_{W'} \\ CF &= CF_W \cup CF_{W'}[Env/a] \\ DF &= DF_W \cup DF_{W'}[Env/a] \\ U &= U_W \cup U_{W'} \end{aligned}$$

When embedding a subprocess into a workflow, we use the communication events with the environment as the interface between the parent and the sub-workflow by wiring *Env* to the proxy component in the parent workflow. In the parent workflow, we replace the component LTS_a by the proxy component $Proxy(a, W')$ that forms the bridge between the two processes, passing on control flow triggers and data items. Its behavior is defined formally in Appendix A.

In the subprocess $LTS_{W'}$, we replace *Env* in communication events by the name a using the substitution

$$\pi_a(e) = \begin{cases} Recv_b(a, msg) & \text{if } e = Recv_b(Env, msg) \\ Send_b(a, msg) & \text{if } e = Send_b(Env, msg) \\ e & \text{otherwise} \end{cases}$$

The overall workflow system is constructed as the composition of the parent workflow without activity a, the communication platform P_W, the proxy, and the subprocess (re-wired to communicate with the proxy as its environment), where $\|$ denotes the synchronized parallel composition operator of Definition 5.3:

$$LTS_{W[a \leftarrow W']} = \left(\|_{a' \in \mathcal{A}_W \setminus \{a\}} LTS_{a'}\right) \| \; LTS_{P_W} \; \| \; LTS_{Proxy(a,W')} \; \| \; LTS_{W'}[\pi_a]$$

The proxy component replaces activity a in the parent workflow and forms the bridge between the parent and the subprocess (or more precisely, its communication platform).

5.6.2. Security of Workflows with Embedded Subprocesses

We now consider the security of a refined workflow $W[a \leftarrow W']$. We assume that policies $\mathcal{P}$ and $\mathcal{P}'$ are given for W and W', respectively, that coincide on the classifications of shared data items and users. Moreover, we assume that the classification of a in $\mathcal{P}$ coincides with the classification of the environment in $\mathcal{P}'$, since the subprocess W' executes in the context of a from the perspective of the parent workflow. For simplicity, we also assume that the sets of domains and the flow relations of $\mathcal{P}$

and $\mathcal{P}'$ are equal. It should be straightforward to relax this assumption, for example, by allowing the domains of the parent workflow to form a sublattice of those of the refinement, so that the internal data items of the subprocess can be classified in a more fine-grained way.

The refined policy $\mathcal{P}[a \leftarrow \mathcal{P}']$ is simply defined as the union of the classification assignments of $\mathcal{P}$ and $\mathcal{P}'$.

Definition 5.9. Let W be a workflow, and let W' be a valid refinement of $a \in \mathcal{A}$ in W. Let $\mathcal{P}$ be a policy on W w.r.t. $\mathcal{D}$. The policy $\mathcal{P}'$ on W' w.r.t. $\mathcal{D}$ is a *valid refinement of $\mathcal{P}$ for a in W* iff

- $ocl_{\mathcal{P}'}(Env) = ocl_{\mathcal{P}}(a)$
- $dom_{\mathcal{P}'}(i) = dom_{\mathcal{P}}(i)$ for all $i \in Docs_W \cap Docs_{W'}$
- $ucl_{\mathcal{P}'}(u) = ucl_{\mathcal{P}}(u)$ for all $u \in U_W \cap U_{W'}$
- the flow relations $\rightsquigarrow_{\mathcal{P}'}$ and $\rightsquigarrow_{\mathcal{P}}$ are equal

The refined policy is defined as $\mathcal{P}[a \leftarrow \mathcal{P}'] = (dom, cl, ocl, ucl, \rightsquigarrow_{\mathcal{P}})$ with

$$dom(a') = \begin{cases} dom_{\mathcal{P}}(a') & \text{if } a' \in \mathcal{A}_W \\ dom_{\mathcal{P}'}(a') & \text{if } a' \in \mathcal{A}_{W'} \end{cases}$$

and analogously for cl, ocl, and ucl.

We assume that the security of the parent workflow W w.r.t. $\mathcal{P}$ has been verified using the compositional approach described above. This allows us to compose the security guarantees of all activities in the parent workflow *except* that of a with the security guarantee of W'. We also assume that W' has been verified w.r.t. $\mathcal{P}'$, although not necessarily using the compositional approach. Its security guarantee is plugged into that of the parent workflow as it is. The bridge between the two workflows is formed by the proxy activity. Intuitively, it declassifies only the information that inputs and outputs of a are mapped to inputs and outputs of W', i.e. they are forwarded to and from the entry and exit activities of W', respectively. Formally, this is captured in a bound $B_{Proxy(a,W')}$ and becomes part of the security guarantee of the refined workflow as follows.

Theorem 5.2. *Let W' be a valid refinement of a in W, and let the policy $\mathcal{P}'$ on W' be a valid refinement of $\mathcal{P}$ for a in W. Let $(LTS_b)_{b \in \mathcal{A}_W}$ be the family of systems defining the behaviors of the activities in W, and let $LTS_{W'}$ be a workflow system for W'. Let*

- *$(\mathcal{V}_b)_{b \in \mathcal{A}_W}$ denote the local views on the activities in W for d w.r.t. $\mathcal{P}$,*
- *$\mathcal{V}_{W'}$ denote the global view on W' and for d w.r.t. $\mathcal{P}$, and*
- *$\mathcal{V}_{W[a \leftarrow W']}$ denote the global view on $W[a \leftarrow W']$ for d w.r.t. $\mathcal{P}[a \leftarrow \mathcal{P}']$.*

If

- $LTS_{a'}$ *satisfies BD Security w.r.t.* $\mathcal{V}_{a'}$ *and* $B_{a'}$ *for all* $a' \in \mathcal{A}$,
- $LTS_{W'}$ *satisfies BD Security w.r.t.* $\mathcal{V}_{W'}$ *and* $B_{W'}$,

then $LTS_{W[a \leftarrow W']}$ *satisfies BD Security w.r.t.* $\mathcal{V}_{W[a \leftarrow W']}$ *and*

$$\begin{aligned} B_{W[a \leftarrow W']} = \big\{ (sl, sl') \mid \forall a' \in \mathcal{A}_W \setminus \{a\}. (sl \restriction Ev_{a'}, sl' \restriction Ev_{a'}) \in B_{a'} \wedge \\ (\pi_a^{-1}(sl \restriction Ev_{W'}), \pi_a^{-1}(sl' \restriction Ev_{W'})) \in B_{W'} \wedge \\ (sl \restriction Ev_a, sl' \restriction Ev_a) \in B_{Proxy(a,W')} \wedge \\ causal(sl') \wedge sl \equiv_{Ev^{obs}_{W[a \leftarrow W']}} sl' \big\} \end{aligned}$$

Since we use the strengthened local and global views from Definition 5.7, there are no additional side conditions to verify for composing the activities of W, the proxy for a, and the workflow system for W'. The resulting bound combines those of the activities in W with that of W', connected by the proxy. As usual, its definition is quite technical, so we will want to simplify it using concrete knowledge about the specific workflows at hand. As an example, we consider the refinement hierarchy of the hiring workflow presented in Section 5.1.

5.6.3. Example

We start with the lowest level of our refinement hierarchy. The medical examination subprocess (Figure 5.3) declassifies nothing about confidential information, beyond the static knowledge that its communication patterns follow the workflow specification. We capture the latter in a predicate $static_{W_3}$.

$$B_{W_3} = \left\{ (sl, sl') \mid static_{W_3}(sl') \wedge sl \equiv_{Ev^{obs}_{W_3}} sl' \right\}$$

In particular, this bound declassifies *nothing* about the content of the medical report, or the occurrence of additional specific examinations.

This is embedded into the abstract medical subprocess, where additionally a medical statement is prepared for the HR department, based on the medical report. This actually declassifies information to observers in that department, as discussed above: it releases the equivalence class of the medical report. To verify this refined workflow $W_2' = W_2[Exam \leftarrow W_3]$, we instantiate Theorem 5.2. By plugging B_{W_2} and B_{W_3} (see above) into the formula for $B_{W_2[Exam \leftarrow W_3]}$ and simplifying it, we obtain the bound

$$\begin{aligned} B_{W_2'} = \big\{ (sl, sl') \mid (sl \restriction Ev_{Report}) \sim_{Med} (sl' \restriction Ev_{Report}) \wedge \\ static_{W_2'}(sl') \wedge sl \equiv_{Ev^{obs}_{W_2'}} sl' \big\} \end{aligned}$$

where

$$static_{W_2'}(sl') = static_{W_3}(\pi^{-1}_{Exam}(sl') \restriction Ev_{W_3}) \wedge static_{Stmt}(sl' \restriction Ev_{Stmt}) \wedge causal(sl')$$

The predicate $static_{W_2'}(sl')$ collects the static knowledge about W_2, W_3, and their composition via the proxy, namely that

- in the subprocess W_3, the occurrence of specific medical diagnoses depends on the result of the initial assessment (which is *not* declassified by W_3),
- all activities (those in W_3 as well as "Prepare statement" and the proxy for the examination) follow well-defined communication protocols, i.e. first receiving input, executing only after being triggered, and only afterwards sending output data to successor activities, followed by triggers,
- communication is causal, including the communication of W_2 with W_3 via the proxy, i.e. each received message corresponds to one message that has been sent before.

Moreover, the equivalence class of the medical report is declassified.[4] However, the occurrence of additional, specific examination activities is still kept secret, as in B_{W_3} above.

Finally, embedding this into the hiring workflow does not add further declassification:

$$B_{W_1'} = \Big\{(sl, sl') \mid (sl \restriction Ev_{Report}) \sim_{Med} (sl' \restriction Ev_{Report}) \wedge static_{W_2'}(sl' \restriction Ev_{W_2'}) \wedge sl \equiv_{Ev^{obs}_{W_1'}} sl'\Big\}$$

This follows trivially from Theorem 5.2, since everything in W_1, including the interface to W_2', is completely observable.

5.7. Related Work

In this chapter, we build upon our previous work presented in [BH14a], which is in turn based on the decomposition methodology presented in [HMSS07]. It uses the MAKS framework to formalize security requirements. By porting the technique to BD Security, we can now explicitly specify and verify declassification policies for workflow systems.

Our notion of workflow refinement is related to action refinement techniques, where atomic events in an abstract model are refined to sequences of events. For

[4] More precisely, the bound $B_{W_2[Exam \leftarrow W_3]}$ only declassifies the equivalence class of the *last* version of the report, while $B_{W_2'}$ also declassifies the number and equivalence classes of *updates* to the report inside W_3. For simplicity, we omit this additional detail in $B_{W_2'}$. Security w.r.t. the former, stronger, bound implies security w.r.t. the latter bound via the translation theorem of Chapter 4.

example, [Hut06] presents an action refinement technique for the MAKS framework. In order to preserve security under refinement, it restricts the ways visible and confidential events may be refined. In particular, confidential events need to remain atomic, although they may be followed by neutral events in the refinement. Our approach is more flexible in the notion of secret and the policy inside the refinement. For example, when refining the "Examination" activity in W_2 to the subprocess W_3, we added the occurrence of the specific diagnosis activities as a new, non-atomic source of confidential information. The disadvantage is that we have to verify additional side conditions, or stronger security properties in the first place, in order to apply our compositional verification approach.

Early examples for workflow management systems with distributed architectures include [AGK+96, MWW+98, SJKB94]. Later, computing paradigms with a similar spirit have emerged, e.g. service-oriented architectures or cloud computing. We see these techniques and standards as complementary to our work, as they can be used for the implementation of our abstract specifications.

Graphical notations such as BPMN are widely used for workflow specification. BPMN extensions to add security annotations to business process diagrams can be found in [BHLR12, RFMP07, WM10]. Closest to the security requirements considered by us comes the notation proposed in [WM10] that supports both the annotation of documents and process lanes with confidentiality and integrity classifications or clearances, respectively, and the annotation of activities with separation of duty constraints, which we will discuss in Chapter 6.

Several proposals for a formal semantics of workflow specifications can be found in the literature. For example, [WG08] maps BPMN diagrams to CSP processes and describes how the formal semantics can be leveraged to compare and analyze workflow diagrams, e.g. with respect to consistency. It focuses on the control flow and does not model data flows. In [YLGY10], workflows are represented as statements in a workflow description language, which is mapped to a representation as hierarchical state machines. An information flow analysis algorithm is described, but the actual information flow property that it checks is not stated in a declarative, mechanism-independent way. [AL12] represents workflows as Petri nets and describes an approach for information flow analysis. The focus is on keeping the occurrence of tasks confidential, whereas our work focuses on the confidentiality of the data that is processed in the workflow. In [ACPP11] and [SLS06], workflows are formalized as transition systems and model-checking is employed to verify properties specified as LTL formulas. This is suitable to verify safety or liveness properties, whereas the information flow predicates considered by us can be seen as hyperproperties [CS10].

5.8. Conclusion

We have presented an approach to formally model both the behavior of a workflow and the associated security requirements, and described how to verify them

compositionally. We have extended our previous work in [BH14a] (and in turn, the decomposition methodology [HMSS07] it is based on) along two dimensions. First, we added declassification control by porting the technique from the MAKS framework to BD Security. Second, we showed how workflow refinement can be mapped to composition, allowing us to leverage our compositionality results for BD Security.

We have sketched how a simple version of our example workflow can be represented as a composition of instantiations of the activity types specified in Appendix A. We have sketched how a model of our example hiring workflow can be built from subprocesses and simple activities, composing their security guarantees. We believe that this compositional approach can help in scaling up verification techniques for information flow scale to larger workflow systems. However, more work is needed before this approach can actually be applied to realistic systems. Possible directions for future work are tool support for translating a more realistic subset of BPMN to our system model, more convenient policy languages, or automatic verification techniques.

Chapter 6.

Composing Safety and Information Flow Properties

6.1. Introduction

In large, distributed systems that facilitate the collaboration of multiple users there are different types of relevant security requirements. So far, this thesis has focused on confidentiality requirements, which can be captured in terms of information flow properties, using frameworks such as BD Security. Integrity requirements are often treated as dual to confidentiality, at least to some degree [BRS10]. However, there are many other security requirements that are *not* concerned with information flow. For example, separation of duty requires that at least two different persons are involved in a decision process. This is typically used to avoid conflicts of interest: for example, a bank employee must not approve his own loan. Process requirements like this can be modeled as safety properties [AS87].

Due to the well-known refinement paradox, the enforcement of a safety property (by prohibiting system runs violating it) can potentially invalidate possibilistic information flow security: For example, consider a workflow system where a separation of duty constraint between a confidential and a non-confidential activity is enforced. Assume that an attacker can see that a certain user performs the non-confidential activity. The attacker then knows that this person has *not* participated in the confidential activity. This might be an information leak in itself (if anonymity is a concern), and it might lead to other information being leaked, such as information about what sequences of actions could have been performed in the confidential activity, if different users are allowed to perform different actions.

Consider again the hiring workflow (Figure 6.1). It involves medical examinations of job candidates, and the medical details of these examinations are considered confidential information. As an example, we consider two types of separation of duty constraints: We require that the medical examinations must be performed by different persons than the rest of the hiring process due to the need-to-know principle, and we require that there must be two independent medical examinations for each candidate performed by different persons to increase confidence in the combined results of the examinations. The information flows in this example are not entirely trivial, because even though the medical details have to be kept confidential from anyone not involved in the examinations, the final decisions (and only the deci-

sions) must be released to the human resources department so that the workflow can continue. Hence, there is some information flow in the presence of separation of duty constraints, and it is not immediately clear whether there might be subtle interrelations between confidentiality and separation of duty.

Another motivation to formally investigate the compatibility of information flow security and safety properties is to support a certain notion of *refinement* in a stepwise development process. The enforcement of a safety property can indeed be seen as a refinement of the target system, where the set of possible behaviors is reduced by excluding those that violate the property. The technique we discuss in this chapter allows us to

1. first verify the information flow security of a (simplified) system where we don't have to consider safety properties like separation of duty, and

2. afterwards refine the system by enforcing safety properties, while preserving (or refining) the security guarantees of the original system.

Note that this notion of *trace refinement* is different from the architectural refinement that we discussed in the previous chapter. While the latter refines the level of abstraction of the specification by adding more detailed subprocesses, the former refines the behavior of the system on the *same* abstraction level.

There is a lot of existing work on security-preserving refinement (cf. Section 6.6). For example, Mantel [Man01b] proposes an approach where the system is modified in a certain, prescribed way, in order to preserve the validity of a given security proof. However, this can lead to undesired results; in particular, the resulting system might be *empty*, i.e. it will refuse to do anything. In contrast, we propose an approach where control over the system behavior is retained, but the *security property* is adapted instead, factoring in any additional information release due to the enforcement of the safety property. This is possible in a seamless way thanks to the flexibility of BD Security. Of course, undesired results are possible here, as well: In the extreme case, the declassification bound obtained for the refined system becomes the identity, which means that *nothing* is guaranteed about the confidentiality of the secret information. It is essential to check whether the security property obtained after refinement is still adequate for the application scenario at hand.

Our approach is based on compositional reasoning. The key observation is that a safety property can be enforced using an execution monitor that runs in parallel with the target system and inhibits executions that would violate the safety property. The monitor synchronizes with the target system on a set of events that are *relevant* for the safety property, i.e., a set of events that is large enough so that the property can be enforced by monitoring them. For example, in the case of separation of duty, only the user interaction events for the constrained activities are relevant, while all other events in the workflow are irrelevant for the property and can be ignored by the monitor. Our compositionality result for BD Security then gives us sufficient conditions under which we can derive information flow security properties of the monitored system. The declassification bound will be a combination of the

original bound of the target system and any additional declassification performed by the monitor. For example, enforcing separation of duty potentially declassifies certain information about which users could have participated in which activities, in addition to the information that the original system declassifies about the secret inputs. Hence, the refinement of the behavior of the original system (by disallowing certain execution traces) is accompanied by a refinement of the security guarantees (potentially declassifying additional information). The monitored system will satisfy *both* the safety property and the refined information flow property.

As an example, we formulate and analyze a monitor for separation of duty, and identify sufficient conditions under which the enforcement of separation of duty preserves security, in particular if the events assigning users to activities are observable and not confidential.

As an additional example, we consider an asynchronous communication platform. We study the impact of enforcing *ordered delivery*, where messages are received in the order that they are sent.[1] The security guarantees of a set of systems communicating via this platform are preserved, if these guarantees do not *depend* on the possibility of messages being delivered out of order. We formalize this by instantiating our compositionality result for a monitor that enforces ordered delivery.

Note that these results apply not only to a single target system, respectively, but to whole *classes* of target systems that have the corresponding relevant events in their interface. Hence, this approach is particularly useful if the target system is not known precisely in advance, or if the same safety property has to be enforced in many different target systems. This approach therefore complements existing techniques for security preserving trace refinement such as [Man01b], which don't have side conditions, but have to be repeated for every concrete target system. Moreover, our approach is not limited to strict preservation of security properties, but allows for their refinement by factoring in the declassification bound of the monitor.

In previous work, we had discussed this approach in the context of the MAKS framework [BH14b].[2] In this chapter, we lift it to BD Security. The rest of this chapter is structured as follows. In Section 6.2, we recall the notion of safety properties from the literature. In Section 6.3, we present the running example that we use in this chapter, a monitor for separation of duty. Section 6.4 formally describes our approach of using compositionality for the security-preserving enforcement of safety properties. Section 6.5 illustrates the approach with another example, namely the enforcement of ordered delivery in an asynchronous communication platform. Section 6.6 discusses related work and Section 6.7 concludes the chapter.

[1] Delivery is not always guaranteed, i.e. messages may get lost, but not arbitrarily: If a message is delivered, all messages sent before are guaranteed to have been delivered already in the correct order. This corresponds to the reliability guarantees of TCP, for example.

[2] Parts of this chapter are adapted from that paper by permission from Springer Nature.

6.2. Safety Properties

A safety property [AS87] can be captured formally as a set of traces where a "bad thing" never happens, e.g. an unauthorized user performing a privileged system action. Conversely, any *invalid* execution contains a finite prefix β where a bad thing occurs that cannot be subsequently "repaired". The following definition is adapted from [LBW05], which takes into account finite traces, unlike [AS87], which assumes infinite streams.

Definition 6.1. Let Ev be a set of events. The set of traces $P \subseteq Ev^*$ is a *safety property on* Ev iff

$$\forall t \in Ev^*.\ t \notin P \longrightarrow (\exists \beta \leq t.\ \forall t' \geq \beta.\ t' \notin P)$$

A system LTS *satisfies* P iff $[\![LTS]\!] \subseteq P$.

Safety properties can be enforced by execution monitors that run in parallel with a target system, observe its executions, and abort any invalid execution before something bad can happen [AS87, LBW05]. This allows us to reason about the enforcement of safety properties in terms of a composition of a target system and a monitor. Beyond safety, *edit automata* [LBW05] have been shown capable of enforcing other properties, including some liveness properties. Edit automata are more powerful, because they can not only abort executions, but also *modify* them. It would be interesting to investigate whether this kind of monitoring can be captured in terms of a composition operator. We leave this as future work and focus on safety properties in this chapter, for which the notion of composition that we have studied so far is sufficient. As a running example, we focus on separation of duty between two activities in a workflow.

Consider a system that includes several activities to be performed with user interaction, such as our hiring workflow running example. Figure 6.1 depicts a variant of the workflow that includes two independent medical examinations. This increases the confidence in the physical fitness of the candidate for the job in question, in cases where the job has particularly high physical demands.

Let a and a' be two activities in such a workflow between which a separation of duty constraint shall be enforced, for example the two medical examinations in Figure 6.1. This means that these activities have to be performed by different users. In other words, the "bad thing" here happens when a single user performs both activities.

Formally, let Ev_a and $Ev_{a'}$, respectively, denote the sets of events belonging to these activities, let Ev denote the set of all events of the workflow system, let U be a set of users, let Ev_u denote the events of interaction between user $u \in U$ and the system, and let $users\colon Ev^* \to 2^U$ return the set of users participating in the events of a given trace. Separation of duty between a and a' is represented by the set of traces

$$P_{SoD}^{a,a'} = \{t \in Ev^* \mid users(t \restriction Ev_a) \cap users(t \restriction Ev_{a'}) = \emptyset\}$$

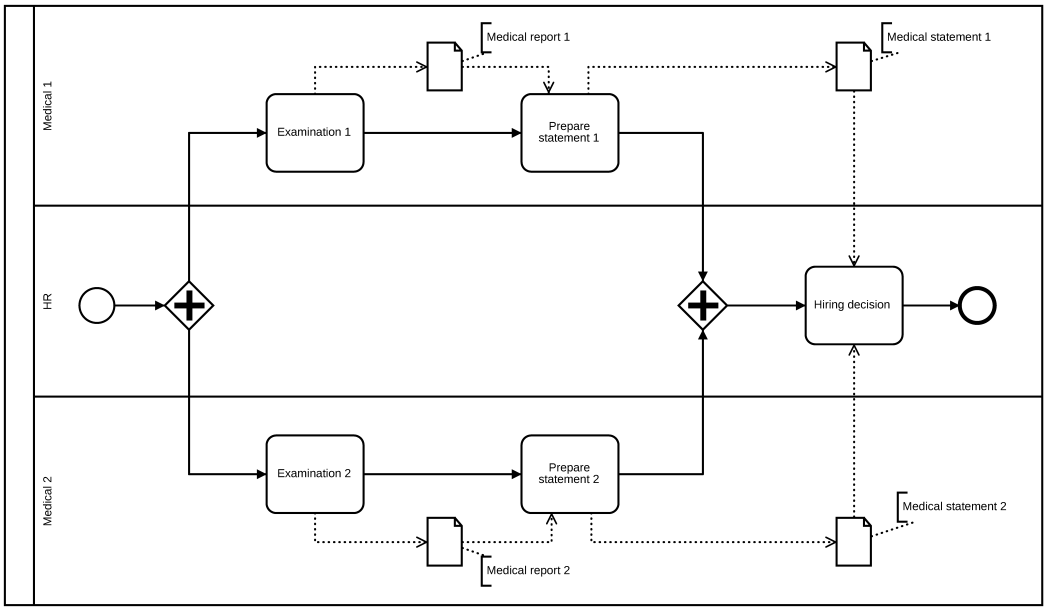

Figure 6.1.: Hiring workflow with two medical examinations

It contains only traces where the users participating in a are different from those participating in a'.

Note that the definition of the above property only depends on the user interaction events occurring in the activities a and a'. Hence, other events are irrelevant for this property and can be ignored by an execution monitor. This is captured formally in the following notion of relevant events:

Definition 6.2. Let P be a safety property on Ev. A set $Ev_P \subseteq Ev$ is a *relevant set of events for* P iff for all $t \in Ev^*$ it holds that $t \restriction Ev_P \in P$ implies $t \in P$.

We will use relevant events as the interface between a monitor and a target system. Note that the above definition of relevant events is not unique: In particular, Ev itself is always a relevant set. We could strengthen the above definition by replacing the word "implies" by "if and only if", which would yield a stronger notion of characteristic events that *precisely* capture the safety property. However, for instantiating our compositionality result, this is not strictly necessary; the implication in one direction is sufficient. It will be prudent in practice, however, to choose the set of relevant events as small as possible in order to keep the side conditions of compositionality tractable: recall that those side conditions mainly refer to the events at the interface between the systems, and the more interface events there are, the stronger the side conditions get. Hence, it is desirable to keep the coupling between target system and monitor loose by keeping the interface small.

We define a monitor simply as a system that satisfies the safety property in question and has a set of events that is relevant for the property.

$$\textsc{Sync}\ \frac{e \in Ev_M \qquad \sigma_T \xrightarrow{e}_T \sigma'_T \qquad \sigma_M \xrightarrow{e}_M \sigma'_M}{(\sigma_T, \sigma_M) \xrightarrow{e} (\sigma'_T, \sigma'_M)} \qquad \textsc{Other}\ \frac{e \notin Ev_M \qquad \sigma_T \xrightarrow{e}_T \sigma'_T}{(\sigma_T, \sigma_M) \xrightarrow{e} (\sigma'_T, \sigma_M)}$$

Figure 6.2.: Transition relation for monitoring composition

Definition 6.3. Let P be a safety property on Ev. A labeled transition system $LTS_M = (St_M, \sigma^0_M, Ev_M, In_M, Out_M, \rightarrow_M)$ *enforces* P iff Ev_M is a relevant set of events for P and $[\![LTS_M]\!] \subseteq P$.

Note that we make the simplifying assumption here that monitors do not have local, internal events of their own that are not relevant for the property: Since P is a safety property on Ev, and Ev_M is a relevant set of events for P, we have $Ev_M \subseteq Ev$. In principle, our results could accommodate internal monitor events, provided that they are neutral (from a security point of view) and disjoint from the events of the property or the target system. However, internal monitor events would complicate the presentation of our results, and in our examples and at the abstraction level that we consider in this chapter, they are not necessary.

In order to enforce a safety property in a target system, we synchronize it with the monitor. The modified transition relation is depicted in Figure 6.2, where $\rightarrow_T$ denotes the transition relation of the target system. The monitored system can only produce an event if the original target system can produce it. In case of events that are relevant for the safety property, we additionally require that the monitor agrees on producing the event. Hence, the monitor can inspect all events in a trace of the target system that are relevant for the property, and abort execution (by refusing to emit an event) if the property is about to be violated.

Definition 6.4. Let Ev_T be a set of events and P be a safety property on Ev_T. Let

$$LTS_T = (St_T, \sigma_T, Ev_T, In_T, Out_T, \rightarrow_T)$$
$$LTS_M = (St_M, \sigma_M, Ev_M, In_M, Out_M, \rightarrow_M)$$

be two labeled transition systems where the latter enforces P. The *monitoring composition of* LTS_T *with* LTS_M, denoted $LTS_T \lhd LTS_M$, is the system

$$LTS_T \lhd LTS_M = (St, \sigma, Ev_T, In_T, Out_T, \rightarrow)$$

where

$$St = St_T \times St_M$$
$$\sigma = (\sigma_T, \sigma_M)$$

and the transition relation $\rightarrow$ is defined by the rules in Figure 6.2.

In the composition $LTS_T \lhd LTS_M$, we will refer to LTS_T as the *target* system, and LTS_M as the *monitor* system. The operator $\lhd$ is very similar to the synchronized

parallel composition operator that we introduced in Definition 5.3. There are two differences. First, we only consider monitors that do not have internal events, as discussed above. Second, we ignore the classification of events of the monitor into inputs and outputs—the two systems simply synchronize on the shared, relevant events. Like the synchronized parallel composition of Chapter 5, the monitoring composition is a special case of the general notion of composition we introduced in Chapter 2, where the synchronization relation is simply the identity on relevant events, and we implicitly perform a translation step after composition that translates the canonical composed events of the form (i, e) and (e_i, e_j) back to the events of the original target system. Again, the latter is possible without affecting either safety or information flow security, because there is a one-to-one mapping between the events of the target and the composed system.

A monitoring composition yields a *refinement* of the target system that satisfies the safety property:

Lemma 6.1. *Let $LTS = (St, \sigma_0, Ev, In, Out, \rightarrow)$ be a system, P be a safety property on Ev, and LTS_M be a system that enforces P. Then $[\![LTS \lhd LTS_M]\!] \subseteq [\![LTS]\!] \cap P$. Hence, $LTS \lhd LTS_M$ satisfies P.*

This follows directly from the above definitions of composition, relevant events, monitor, and set inclusion.

6.3. Example: Separation of Duty

We have seen on page 130 how to formalize separation of duty as a safety property. For enforcing it, we define the monitor $LTS^{a,a'}_{SoD} = (St_{SoD}, \sigma^0_{SoD}, Ev^{a,a'}_{SoD}, \emptyset, \emptyset, \rightarrow)$ with

$$\begin{aligned} St_{SoD} &= 2^U \times 2^U \\ \sigma^0_{SoD} &= \{\emptyset, \emptyset\} \\ Ev^{a,a'}_{SoD} &= (Ev_a \cup Ev_{a'}) \cap Ev_U \end{aligned}$$

where $Ev_U = \bigcup_{u \in U} Ev_u$ is the set of all user interaction events and $\rightarrow$ is the transition relation given in Figure 6.3. The monitor simply keeps track of which users have already participated in the activities a and a', respectively. Hence, the state is a pair of sets of users, initially empty. $LTS^{a,a'}_{SoD}$ monitors the user interaction events of the two activities, and only allows an event in which user u participates in a if u has not participated in a' before (and vice versa). Every time an additional user participates in one of the activities, this is recorded in the state.

It is easy to see that $Ev^{a,a'}_{SoD}$ is a relevant set of events and $LTS^{a,a'}_{SoD}$ is a monitor for $P^{a,a'}_{SoD}$: it prevents the "bad thing" of one user participating in both activities. By Lemma 6.1, composing this monitor with a target system involving activities a and a' and users U enforces separation of duty between a and a'.

However, BD security properties of the target system are not preserved in general upon composition with the monitor. As discussed already, enforcing separation of

$$\textsc{Act}_a \ \frac{e \in Ev_a \setminus Ev_{a'} \qquad users(e) \cap \sigma_{a'} = \emptyset}{(\sigma_a, \sigma_{a'}) \xrightarrow{e} (\sigma_a \cup users(e), \sigma_{a'})}$$

$$\textsc{Act}_{a'} \ \frac{e \in Ev_{a'} \setminus Ev_a \qquad users(e) \cap \sigma_a = \emptyset}{(\sigma_a, \sigma_{a'}) \xrightarrow{e} (\sigma_a, \sigma_{a'} \cup users(e))}$$

Figure 6.3.: Transition relation for a separation of duty monitor

duty might reveal information about the user assignment to the attacker, which might in turn leak secret information: Consider, for example, security requirements involving anonymity of users, or scenarios where certain secret events can only be performed by certain users.

In these cases, the security property of the monitored system will have to factor in this additional information release in the declassification bound. Intuitively, enforcing separation of duty between the activities a and a' gives the attacker the additional knowledge that any user who participated in a did not participate in a', and vice versa.

Let us discuss how we can use our compositionality framework to formally derive a refined security property for the monitored system. Consider some workflow system LTS_W that we have verified for confidentiality using the approach of Chapter 5. Hence, we have proved that it satisfies a BD Security property w.r.t. a bound B and the global workflow view $\mathcal{V}_{\mathcal{A}}^{d}$ for each security domain d (cf. Definition 5.7). Let us assume that we now additionally want to enforce a separation of duty constraint between two activities a and a' using the monitor $LTS_{SoD}^{a,a'}$. In order to formulate a BD Security property of the monitor that we can compose with that of the target system, we have to define a view and a bound for the monitor. Since we want to know whether and to what degree the security property of the target system is preserved when enforcing the safety property, we have to use a view for the monitor that aligns with that of the target system. For this example, we simply mirror the view of the target system from Definition 5.7, restricted to the relevant events of the monitor.

$$\mathcal{V}_{SoD}^{a,a'} = \left(Ev_{\mathcal{A}}^{obs} \cap Ev_{SoD}^{a,a'}, \mathrm{id}, Ev_{\mathcal{A}}^{sec} \cap Ev_{SoD}^{a,a'}, \mathrm{id}\right)$$

Note that the secret events for the monitor are in fact *all* events in $Ev_{SoD}^{a,a'}$. Recall that in the global workflow views of Definition 5.7, we considered *all* input and output events in the workflow to be *potentially* secret. While typically only few events genuinely contain secret information, their occurrence might *depend* on non-confidential events. For example, a secret input to an activity can only occur after the activity has been started via a *Start* event. Treating all of these events as potentially secret allows us to refer to them in the declassification bound and specify

the dependencies between them and genuinely secret events explicitly, as we have done in the examples on page 119. These bounds require the arbitrary insertion, deletion, and replacement of secret input events, but only in positions allowed by predicates capturing the static knowledge of an observer about the dependencies of events, e.g. no secret inputs before the *Start* event. Those predicates depend on non-confidential events, but the bounds do not require anything about the non-confidential events themselves.

For the bound for the separation of duty monitor, it is actually necessary that all relevant events are treated as secret to some degree (if *any* one of them is secret). Indeed, whether any relevant event is possible at a given position in a trace depends on *all* relevant events that have happened before, and this needs to be specified in the bound.

From an attacker perspective, a separation of duty constraint between activities a and a' gives observers the static knowledge that those activities are performed by different users. We formalize this in the following bound for the monitor.

$$\begin{aligned} B_{SoD}^{a,a'} &= \left\{(sl, sl') \mid sl' \in P_{SoD}^{a,a'} \wedge sl =_{Ev_{\mathcal{A}}^{obs}} sl'\right\} \\ &= \left\{(sl, sl') \mid \left(\forall u \in U.\ u \in \mathrm{users}(sl' \upharpoonright Ev_a) \longrightarrow u \notin \mathrm{users}(sl' \upharpoonright Ev_{a'})\right) \wedge \right. \\ &\qquad \left. sl =_{Ev_{\mathcal{A}}^{obs}} sl'\right\} \end{aligned}$$

It is trivial to prove that the monitor $LTS_{SoD}^{a,a'}$ satisfies BD Security w.r.t. $\mathcal{V}_{SoD}^{a,a'}$ and $B_{SoD}^{a,a'}$. The monitor can produce exactly the set of traces in $P_{SoD}^{a,a'}$ (restricted to relevant events), in particular the sequence of events sl' required by the bound.

The compositionality side conditions of Lemma 3.3 are trivially satisfied as well. There are no neutral events at the interface between the workflow system and the monitor, the secrets and observations are fully concrete events, and the scheduling of secret and observable events is specified in the bound of the workflow system.

Hence, we can apply our compositionality result to show that the monitored system $LTS_W \triangleleft LTS_{SoD}^{a,a'}$ satisfies a BD Security with the merged declassification bound

$$B' = \left\{(sl, sl') \in B \mid \forall u \in U.\ u \in \mathrm{users}(sl' \upharpoonright Ev_a) \longrightarrow u \notin \mathrm{users}(sl' \upharpoonright Ev_{a'})\right\}$$

This bound is a refinement of the original bound B. In addition to the information declassified by the original target system, it releases the information that any user who has participated in the observable activity a has *not* participated in the secret activity a'. Whether or not this is acceptable depends on the application scenario. As always when modifying a declassification bound, one has to carefully check whether the newly obtained bound still adequately captures the security requirements. In our hiring workflow, for example, the additional information release is unproblematic: After all, we only consider the content of medical reports to be confidential, not the identities of the persons who created it.[3]

[3]This assumes that there is no (deterministic) dependency of the secret information (i.e., diag-

6.4. Secure Monitoring of Safety Properties

We now formulate this compositional approach to securely enforcing safety properties in more general terms. In particular, we want to facilitate reusability of secure monitors: verifying *once* that they provide suitable security guarantees, so that they can be composed securely with *many* different target systems. For example, the separation of duty monitor $LTS_{SoD}^{a,a'}$ can be composed with any workflow system that includes the activities a and a'. Whether the safety monitor can provide a *security* guarantee, and if so, which one, depends on the security property of the target system. For the separation of duty monitor, for example, the compositional approach can only be used for target systems with security properties that do not have neutral events at the interface with the monitor, i.e., all user interaction events of the activities a and a' are either observable or (potentially) secret. Otherwise, the side conditions of compositionality would require us to prove that the monitor can always accept any neutral event, say, $Start_a(u)$, which violates separation of duty if the user u is already participating in the other activity a'. Other monitors have other constraints: The monitor for causal delivery that we discuss in Section 6.5 *does* support neutral events, but has other, more fine-grained constraints on the views of target systems. We will capture such constraints formally by modeling the security guarantees of a monitor as a function of the view of the target system. We only define this function for *supported* target views for which we can verify a corresponding security guarantee of the monitor. For unsupported target system views, the security guarantee of the monitor is undefined.

Assuming we have a supported target view, the next question is which view to use for the monitor. As we have seen in the separation of duty example, the views of monitor and target system need to be aligned very closely in order for compositional reasoning to be applicable. In particular, the views have to agree on the attacker models, i.e., which events at the interface are observable. Hence, we choose the same set of observable events and the same observation extracting function for the monitor as for the target system, restricted to relevant events. For the secrets, however, it turns out that it is useful to use a more fine-grained notion: we always assume the *identity* as the secret extracting function for the monitor. Using the events themselves as secrets instead of abstracting away information results in a stronger security property of the monitor that is easier to compose. Indeed, choosing anything other than the identity here would lead to proof obligations on the side of the *target* system during composition, in order to show that the details of secrets we abstract away during the verification of the monitor do not matter for the target system's security property, either. The guiding principle we use in this chapter is to shift as many proof obligations as possible to the side of the monitor. The rationale behind this is twofold:

nosed diseases) on the identity of the examiner. In scenarios where a difference in likelihood of diagnoses by different persons is a serious concern, it might be worth considering probabilistic models of security as an alternative to BD Security.

- The monitor is typically the smaller and simpler system, so the side conditions are likely to be easier to verify for the monitor.
- The stronger the security guarantees of the monitor are, the more target systems can be composed with it without additional proof effort. This supports our aim of *reusable* monitors: verified once, composable with many target systems.

In addition to choosing a strong notion of secret for the monitor, this guiding principle will also lead us to impose stricter requirements regarding the acceptance of neutral events on the monitor rather than on the target system. Nevertheless, we leave one particular degree of freedom for choosing the monitor view: We allow some events that are secret for the target system to be treated as neutral for the monitor. This is important if the monitor cannot make any security guarantees about those events. We didn't need this flexibility in the separation of duty example, but we will make use of it in the example of Section 6.5. However, neutral events on the side of the monitor should only be used if necessary for verification: It adds proof obligations for the target system, and it weakens the overall security guarantee of the composed system, because events that are neutral for the monitor will become neutral in the composition as well.

We now proceed to state this approach formally. As discussed above, whether we can compose a monitor with a target system and, if so, what the resulting security property is, depends on the original security property of the target system in question, in particular on the view. Hence, we specify the security guarantee of a monitor as a function G from a set of supported target system views to corresponding security properties of the monitor. Since most of the parameters of that security property are constrained by the target system property and the side conditions of compositionality, two parameters suffice to specify the security property of the monitor: the set of events that have to be treated as neutral for the monitor, and the declassification bound of the monitor. If $G(\mathcal{V}) = (N_M, B_M)$ for a target system view $\mathcal{V}$, then this claims that the monitor satisfies a BD Security property with the bound B_M and a view that mirrors $\mathcal{V}$, but with the events in N_M treated as neutral.

Definition 6.5. Let Ev_M be the set of events of a monitor LTS_M, and let $\mathcal{V}\mathbf{s}$ be a set of target system views on sets of events that include Ev_M. A *monitor security guarantee for* $\mathcal{V}\mathbf{s}$ *on* Ev_M is a function

$$G\colon \mathcal{V}\mathbf{s} \to 2^{Ev_M} \times 2^{(Ev_M)^* \times (Ev_M)^*}$$

from target system views to sets of neutral events in Ev_M and declassification bounds on Ev_M.

G induces a function $\mathcal{V}_M$ from $\mathcal{V}\mathbf{s}$ to monitor views, called the *view mapping for* G. We define it for any target system view $\mathcal{V} \in \mathcal{V}\mathbf{s}$ with $G(\mathcal{V}) = (N_M, B_M)$ as

$$\mathcal{V}_M(\mathcal{V}) \equiv \Big(Ev_{\mathcal{V}}^{obs}, getObs_{\mathcal{V}}, (Ev_{\mathcal{V}}^{sec} \cup N_{\mathcal{V}}) \cap (Ev_M \setminus N_M), \mathrm{id}\Big)$$

where $N_{\mathcal{V}} = Ev_M \setminus (Ev_{\mathcal{V}}^{obs} \cup Ev_{\mathcal{V}}^{sec})$ is the set of monitor events that are neutral for the target system.

For example, the separation of duty monitor of Section 6.3 does not require any neutral events, and it declassifies the static knowledge that activities a and a' are performed by different users, as captured in the bound $B_{SoD}^{a,a'}$ on page 135. Hence, we can choose the set of all workflow views $\mathcal{V}_{\mathcal{A}}^{d}$ with $a, a' \in \mathcal{A}$ as the set of supported views $\mathcal{V}\mathbf{s}$ and set $G(\mathcal{V}_{\mathcal{A}}^{d}) = (\emptyset, B_{SoD}^{a,a'})$.

Intuitively, a monitor *provides* such a security guarantee G if, for all target system views $\mathcal{V} \in \mathcal{V}\mathbf{s}$ with $G(\mathcal{V}) = (N_M, B_M)$, it satisfies BD Security w.r.t. $\mathcal{V}_M(\mathcal{V})$ and B_M. Additionally, we require it to satisfy its part of the side conditions for compositionality as laid out in Lemma 3.3. Essentially, we have to prove that the security guarantee of the monitor does not depend on

- the (non-)occurrence of neutral events of the target system, and
- any information that is abstracted away from secret and observable events in the view on the target system.

Recall that in Lemma 3.3 we have formalized these requirements as security properties involving the arbitrary insertion of neutral events and the local replacement of interface events that correspond to indistinguishable events in the view on the other system. We reformulate these requirements in the context of monitor composition as follows.

Definition 6.6. Let $LTS_M = (St_M, \sigma_M, Ev_M, In_M, Out_M, \rightarrow_M)$ be a monitor, let G be a monitor security guarantee for $\mathcal{V}\mathbf{s}$ on Ev_M, and let $\mathcal{V}_M$ be the view mapping for G.

LTS_M *provides* G iff, for all $\mathcal{V} \in \mathcal{V}\mathbf{s}$ with $G(\mathcal{V}) = (N_M, B_M)$,

- it satisfies BD Security w.r.t. $\mathcal{V}_M(\mathcal{V})$ and B_M,
- it accepts the neutral events of $\mathcal{V}$ via Id_{Ev_M} in $\mathcal{V}_M(\mathcal{V})$,
- if $N_{\mathcal{V}} \neq \emptyset$ and $N_M \neq \emptyset$, it silently accepts the neutral events of $\mathcal{V}$ before $Ev_M \setminus (N_{\mathcal{V}} \cup N_M)$ via Id_{Ev_M} in $\mathcal{V}_M(\mathcal{V})$, and
- it supports local replacement via $\approx_{\mathcal{V}}$ for $\mathcal{V}_M(\mathcal{V})$ and B_M.

If $\mathcal{V}\mathbf{s}$ only contains workflow views $\mathcal{V}_{\mathcal{A}}^{d}$, for example, the additional conditions above are trivially satisfied, because there are no neutral events and $\approx_{\mathcal{V}_{\mathcal{A}}^{d}}$ is the identity. Let us discuss which features of the views $\mathcal{V}_{\mathcal{A}}^{d}$ are required for the monitoring approach to work in the case of separation of duty, and which aspects can be relaxed. Having no neutral user interaction events in the activities a and a' is necessary for applying our compositionality result to the separation of duty monitor. Otherwise, the side conditions of compositionality would require us to prove that the monitor accepts neutral events at any time, which is not the case (it might violate separation

of duty). It is also necessary to include observable relevant events in the set of secret events, if any relevant event is secret. As discussed in Section 6.3, this is required to capture the interdependencies of relevant events in the declassification bound: whether any relevant event is possible at a certain point in a trace depends on *all* relevant events that have happened before. Note that this does not rule out observable events, it merely forces us to be more explicit about them, in particular about their scheduling relative to secret events. Indeed, the workflow views $\mathcal{V}^d_{\mathcal{A}}$ treat most events as *both* partially observable and partially secret, and the corresponding bounds specify their interdependencies. One aspect that we *can* relax compared to $\mathcal{V}^d_{\mathcal{A}}$, however, is the notion of secrets and observations that get extracted from events. The separation of duty monitor does not care about the actual contents of the events, it only has to check which user participated in an event and to which activity it belongs. Hence, it is sufficient if the secrets and observations (of the target system) contain enough information to distinguish users and activities. We formalize the notion of a view distinguishing a set of events as follows.

Definition 6.7. A view $\mathcal{V}$ *distinguishes* a set of events E iff $e \approx^{obs}_{\mathcal{V}} e'$ or $e \approx^{sec}_{\mathcal{V}} e'$ implies $(e \in E \longleftrightarrow e' \in E)$.

A view $\mathcal{V}$ is *invertible on* E iff $\approx_{\mathcal{V}}$ is the equality on E.

We can now formulate the following security guarantee for the separation of duty monitor.

Lemma 6.2. *Let* $\mathcal{V}$**s** *be the set of views* $\mathcal{V}$ *where*

- $\mathcal{V}$ *distinguishes* Ev_a, $Ev_{a'}$, Ev_u *for all* $u \in U$, *and* $Ev^{sec}_{\mathcal{V}} \cap Ev^{obs}_{\mathcal{V}}$, *and*
- *one of the following holds:*
  - $Ev^{a,a'}_{SoD} \subseteq Ev^{sec}_{\mathcal{V}}$, *or*
  - $Ev^{a,a'}_{SoD} \subseteq Ev^{obs}_{\mathcal{V}} \setminus Ev^{sec}_{\mathcal{V}}$ *and* $\mathcal{V}$ *is invertible on* $Ev^{a,a'}_{SoD}$.

Let $G^{a,a'}_{SoD}$ *be the security guarantee on* $\mathcal{V}$**s** *with* $G^{a,a'}_{SoD}(\mathcal{V}) = (\emptyset, B^{a,a'}_{SoD}(\mathcal{V}))$, *where*

$$B^{a,a'}_{SoD}(\mathcal{V}) = \left\{(sl, sl') \mid sl' \in P^{a,a'}_{SoD} \wedge O_{\mathcal{V}}(sl) = O_{\mathcal{V}}(sl')\right\}$$

The monitor $LTS^{a,a'}_{SoD}$ *provides* $G^{a,a'}_{SoD}$.

In addition to workflow views of the form $\mathcal{V}^d_{\mathcal{A}}$, this allows views with more abstract secrets and observations, provided that they contain sufficient information to distinguish activities and users. In addition to the case where all relevant events are treated as partially secret, it also supports views where both activities are purely observable and do not handle secret information. In that case, enforcing separation of duty trivially preserves the security property of the target system.

In order to be able in general to compose such a monitor and its security guarantee with a target system, we make a few more assumptions on the views and bounds to

satisfy the side conditions of compositionality. In particular, we generally assume that the relevant events are either all secret to some degree, or all observable. As discussed above, this implies that the *ordering* of secrets and observations is fully reflected in the sequence of secrets (or the sequence of observations, respectively). It is possible to relax this assumption, but then Lemma 3.3 would require us to prove that *both* the target system and the monitor support flexible scheduling of secrets and observations. By strengthening the views, we avoid this additional proof obligation for the target system.

Moreover, we make a few more well-definedness assumptions to ensure composability of the monitor. For example, if we declare a set of events N_M to be neutral for the monitor, then those events must be secret in the target system view, since we will formalize the requirement that the target system always accepts those events as a BD Security property, as usual. Moreover, the monitor bound must be total in events that are neutral for the target system, and support replacement of events that are view-equivalent for the target system.

Definition 6.8. A monitor security guarantee G for $\mathcal{V}\mathbf{s}$ on Ev_M is *composable* iff, for all $\mathcal{V} \in \mathcal{V}\mathbf{s}$ with $G(\mathcal{V}) = (N_M, B_M)$, all of the following hold:

- Either $Ev_{\mathcal{V}}^{obs} \cap Ev_M \subseteq Ev_{\mathcal{V}}^{sec}$ or $(Ev_{\mathcal{V}}^{sec} \cap Ev_M) \setminus N_M \subseteq Ev_{\mathcal{V}}^{obs}$ holds.
- $N_M \subseteq Ev_{\mathcal{V}}^{sec} \setminus Ev_{\mathcal{V}}^{obs}$ holds.
- B_M includes $\approx_{\mathcal{V}}$ and is total in $N_{\mathcal{V}}$.
- $\mathcal{V}$ distinguishes Ev_M, N_M, and $Ev_{\mathcal{V}}^{sec} \cap Ev_{\mathcal{V}}^{obs}$.
- $\mathcal{V}$ is invertible on N_M and on $Ev_M \setminus Ev_{\mathcal{V}}^{sec}$.

Once we have verified that a monitor provides a composable security guarantee, we can compose it with any target system that is secure w.r.t. a view $\mathcal{V} \in \mathcal{V}\mathbf{s}$ that is supported by the monitor, obtaining a system that satisfies both the safety property and a refined security property.

Theorem 6.3. *Let $LTS = (St, \sigma, Ev, In, Out, \rightarrow)$ be a system that satisfies BD Security w.r.t. a view $\mathcal{V} \in \mathcal{V}\mathbf{s}$ and a bound B. Let LTS_M be a monitor that provides a composable security guarantee G for $\mathcal{V}\mathbf{s}$ on Ev_M, where $G(\mathcal{V}) = (N_M, B_M)$. Let $\mathcal{V}_M$ be the view mapping for G. If either $N_M = \emptyset$ or B is total in N_M and LTS supports eager insertion of N_M for $\mathcal{V}$ and B, then $LTS \triangleleft LTS_M$ satisfies BD Security w.r.t. $\mathcal{V}'$ and B', where*

$$\mathcal{V}' = \Big(Ev_{\mathcal{V}}^{obs}, getObs_{\mathcal{V}}, Ev_{\mathcal{V}}^{sec} \setminus N_M, getSec_{\mathcal{V}}\Big)$$

$$\begin{aligned} B' = \big\{(sl, sl') \in B \mid\ & sl \preceq_C sl' \wedge \\ & (\forall t \in Ev^*.\ S_{\mathcal{V}'}(t) = sl \longrightarrow \\ & \quad (\exists t' \in Ev^*.\ S_{\mathcal{V}'}(t') = sl' \wedge \\ & \quad\quad (t \restriction Ev_{\mathcal{V}_M(\mathcal{V})}^{sec}, t' \restriction Ev_{\mathcal{V}_M(\mathcal{V})}^{sec}) \in B_M))\big\} \end{aligned}$$

The security property is weakened by making the events in N_M neutral, and the bound is refined to include the declassification performed by the monitor: Only those sequences of secrets are guaranteed to be indistinguishable for an observer where also the monitor can guarantee that the subsequences of relevant secret events are indistinguishable. The definition of B' is complicated by the fact that the notion of secrets used in the target system and the monitor may be different, and we need to take into account both of them. The monitor is guaranteed to have the stronger notion of secrets, since we assume that it uses the events themselves as the secrets. Hence, for any pair of secret sequences, B' checks whether the abstract secrets of the target system can be produced by *some* sequence of secret events supported by the bound of the monitor. As with our general compositionality result, the modified bound intuitively declassifies the combination of information declassified by the component bounds. For example, when enforcing separation of duty with the monitor of Section 6.3, the information that gets declassified is that of the original target system combined with the knowledge that the activities a and a' are performed by different users. If the target system uses the same notion of secrets as the monitor (i.e., full events), then the definition of B' can be simplified to a kind of intersection of the bounds; this is the case for the workflow bounds $\mathcal{V}^{a,a'}_{SoD}$, and we have already seen how the composed bound B' looks like in Section 6.3.

Note that, since we have taken care to shift as many of the side conditions for compositionality towards the monitor, the only remaining side condition for the target system is that the neutral monitor events can be deleted and inserted arbitrarily in a backwards-strict way, if N_M is non-empty. No other side conditions are required for the target system.

Nevertheless, if the target system *does* satisfy a side condition (such as local replacement) that is also satisfied by the monitor, this side condition is preserved under the constraints laid out in Theorem 3.7. Note that the refined bound B' after composition with the monitor is guaranteed to be a backwards-strict composition in the sense of Definition 3.25, i.e. $B' \subseteq B \parallel^{BS}_{B} B_M$ holds.[4] Hence, it only remains to consult Table 3.2 for the preservation of side conditions.

As a trivial example for an application of Theorem 6.3, consider the enforcement of a safety property where all relevant events are purely and completely observable. In this case, the monitor does not constrain secret events at all, and the security property of the target system is preserved as it is.

Lemma 6.4. *Any monitor LTS_M with the set of events Ev_M provides the monitor security guarantee G^{obs} for $\mathcal{V}\mathbf{s}^{\text{obs}}$ on Ev_M, where $G^{\text{obs}}(\mathcal{V}) = (\emptyset, Id)$ for all $\mathcal{V} \in \mathcal{V}\mathbf{s}^{\text{obs}}$, and $\mathcal{V}\mathbf{s}^{\text{obs}}$ is the set of views $\mathcal{V}$ where $Ev_M \subseteq Ev^{obs}_{\mathcal{V}} \setminus Ev^{sec}_{\mathcal{V}}$ and $\mathcal{V}$ is invertible on Ev_M and distinguishes Ev_M as well as $Ev^{sec}_{\mathcal{V}} \cap Ev^{obs}_{\mathcal{V}}$.*

For any target system LTS that is BD secure w.r.t. a view $\mathcal{V} \in \mathcal{V}\mathbf{s}^{\text{obs}}$ and a bound B, the monitored system $LTS \lhd LTS_M$ is still BD secure w.r.t. $\mathcal{V}$ and B.

[4] This is due to the fact that, on the one hand, the (de)composition of secrets here is unique up to $\approx_{\mathcal{V}}$ and neutral events, because $\parallel$ is the identity, and, on the other hand, B_M includes $\approx_{\mathcal{V}}$ and both bounds are total in neutral events. See Lemma C.2 for a proof sketch.

Even though this lemma may already be useful for the refinement of the purely observable behavior of a target system, our framework is not limited to such extreme cases. The refinement of a workflow bound that we derived in Section 6.3 can be obtained by instantiating Theorem 6.3, and in the following section, we discuss another example that additionally involves neutral events on the side of the monitor.

6.5. Example: Ordered Delivery of Asynchronous Messages

While working on [BH14a], we encountered the guarantee of ordered delivery of messages by an asynchronous communication platform as a safety property that we wanted to enforce in workflow systems. When we initially specified our workflow system in terms of communicating subsystems in [BH14a], we did not include any guarantees regarding message delivery in the specification of the communication platform. This simplified both the specification of the platform and the proof of compositionality, but it made the specifications of the communicating subsystems more complex. We had to introduce explicit acknowledgment messages and let the subsystems wait for acknowledgments before continuing with a communication protocol in some cases. Ordered message delivery per sender-receiver pair, i.e., the guarantee that messages between two components are received in the order that they are sent, makes these explicit acknowledgments unnecessary in the cases we encountered. However, the question arises if this refinement of the communication platform preserves the security guarantees that we had verified for the workflow system. It turns out that we can use the same compositional approach as for separation of duty above to analyze the impact of enforcing ordered delivery on the information flow security properties of the workflow system.

We first formulate ordered delivery as a safety property. Let $\mathcal{A}$ be a set of agents exchanging messages $m \in \mathcal{M}$ via the communication platform, represented by events of the form $Send_a(b, m)$ or $Recv_b(a, m)$. Let $sentMsgs(a, b, t)$ and $rcvdMsgs(b, a, t)$ denote the sequences of messages m sent from a to b, or received by b from a, respectively, in a trace t, and let $\leq$ be the prefix order on sequences. We define ordered delivery as

$$P_{OD} = \left\{ t \mid \forall t' \leq t.\ \forall a, b.\ rcvdMsgs(b, a, t') \leq sentMsgs(a, b, t') \right\}$$

We define a monitor for this property with the relevant set of events

$$Ev_{OD} = \{e \mid \exists a, b, m.\ e = Send_a(b, m) \vee e = Recv_b(a, m)\}$$

For each pair of agents, the state of the monitor records the ordered sequence of messages currently "in transit". Formally, $LTS_{OD} = (St_{OD}, \sigma^0_{OD}, Ev_{OD}, \emptyset, \emptyset, \rightarrow)$ where

$$St_{OD} = (\mathcal{A} \times \mathcal{A} \to \mathcal{M}^*)$$
$$\sigma^0_{OD} = ((a, b) \mapsto \langle\rangle)$$

and the transition relation $\rightarrow$ is given in Figure 6.4. The monitor acts as a FIFO buffer: Messages can be sent at any time and are appended to the corresponding buffer, but a message may only be received by b from a if it is the oldest message sent from a to b that has not been received already.

$$\textsc{Send}\ \frac{}{\sigma \xrightarrow{\mathrm{Send}_a(b,m)} \sigma((a,b) := \sigma(a,b) \cdot m)}$$

$$\textsc{Recv}\ \frac{\sigma(a,b) = m \cdot ms}{\sigma \xrightarrow{\mathrm{Recv}_b(a,m)} \sigma((a,b) := ms)}$$

Figure 6.4.: Transition relation for a ordered delivery monitor

The security views we had used in [BH14a] allow neutral *Send* events. It turns out that, in order for the refined communication platform to be secure, for each *Send* event that is neutral for the target system, we have to treat the corresponding *Recv* event as neutral for the platform. To see why, consider the situation where the platform has to accept a new message coming from the target system via a neutral *Send* event. If there are further messages on this channel that actually get delivered later in the trace, then the platform has to deliver the newly inserted message after the currently pending ones, but before those coming later. Otherwise, ordered delivery would be violated. Hence, the platform needs the flexibility to react to changes in neutral *Send* events by adapting neutral *Recv* events. Combined with a few other simplifying assumptions and conditions for well-definedness and composability, we arrive at the following definition of a monitor security guarantee for ordered delivery:

Lemma 6.5. *Let $\mathcal{V}\mathbf{s}_{OD}$ be the set of views $\mathcal{V}$ where*

- $Ev_{OD} \cap Ev_{\mathcal{V}}^{obs} \subseteq Ev_{\mathcal{V}}^{sec}$,
- $Send_a(b,m) \in Ev_{\mathcal{V}}^{obs} \longleftrightarrow Recv_b(a,m) \in Ev_{\mathcal{V}}^{obs}$ *for all* a, b, m,
- $Recv_b(a,m) \in Ev_{\mathcal{V}}^{sec}$ *for all* a, b, m, *and*
- $\mathcal{V}$ *is invertible on* Ev_{OD} *and distinguishes* Ev_{OD} *as well as* $Ev_{\mathcal{V}}^{sec} \cap Ev_{\mathcal{V}}^{obs}$.

The monitor LTS_{OD} provides the security guarantee G_{OD} for $\mathcal{V}\mathbf{s}_{OD}$ on Ev_{OD}, where, for any $\mathcal{V} \in \mathcal{V}\mathbf{s}_{OD}$, $G_{OD}(\mathcal{V}) \equiv (N_{OD}, B_{OD})$ with

$$N_{OD} = \left\{ e \mid \exists a, b, m.\ e = Recv_a(b,m) \wedge Send_b(a,m) \notin (Ev_{\mathcal{V}}^{sec} \cup Ev_{\mathcal{V}}^{obs}) \right\}$$
$$B_{OD} = \left\{ (sl, sl') \mid (sl' \setminus N_{\mathcal{V}}) \in P_{OD} \wedge O_{\mathcal{V}}(sl) = O_{\mathcal{V}}(sl') \right\}$$

In contrast to the separation of duty monitor, there is an additional side condition on the target system that follows directly from the requirement of well-behavedness:

The target system must securely accept neutral *Recv* events at any time in order to synchronize with the refined communication platform. Moreover, *Recv* events that correspond to neutral *Send* events become neutral in the security view of the composed system. Pairs of *Send* and *Recv* events that are confidential, however, remain confidential (analogously for observable events).

Theorem 6.6. *Let LTS be a target system and* $\mathcal{V} \in \mathcal{V}\mathbf{s}_{OD}$ *be a view for LTS where* $G_{OD}(\mathcal{V}) = (N_{OD}, B_{OD})$*. Let* B *be a bound that is total in* N_{OD}*. If LTS satisfies BD Security w.r.t.* $\mathcal{V}$ *and* B*, and if LTS supports eager insertion of* N_{OD} *for* $\mathcal{V}$ *and* B*, then* $LTS \triangleleft LTS_{OD}$ *satisfies BD Security w.r.t.*

$$\mathcal{V}' = (Ev^{obs}_{\mathcal{V}}, getObs_{\mathcal{V}}, Ev^{sec}_{\mathcal{V}} \setminus N_{OD}, getSec_{\mathcal{V}})$$
$$B' = \left\{(S_{\mathcal{V}}(t), S_{\mathcal{V}}(t')) \in B \mid (t \restriction (Ev_{OD} \cap Ev^{sec}_{\mathcal{V}'}), t \restriction (Ev_{OD} \cap Ev^{sec}_{\mathcal{V}'})) \in B_{OD}\right\}$$

The refined bound essentially corresponds to an intersection of the target and monitor bounds (modulo the restriction on relevant events): The monitored system additionally declassifies that messages (if they are delivered) are received in the order that they have been sent.

Coming back to our earlier case study [BH14a], we had verified the workflow system using the compositional approach also described in Chapter 5. For this purpose, we had already proved eager insertion of *Recv* events for the workflow system, even before considering ordered delivery. This allows us to apply Theorem 6.6 without further proof obligations. The resulting security property is still adequate for the case study in [BH14a]. Treating *Recv* events as neutral is unproblematic in this scenario, because the secret information originates not in the messages sent between the components of the system, but in the input received from the *users* from the outside (which are modeled by a separate set of events); the internal computations in the activities themselves do not generate confidential information. Moreover, the ordering of messages is not confidential, either. In fact, the communication patterns are determined by the workflow specification, which is assumed to be known in advance. Hence, the refinement preserves the adequacy of the security properties for the scenario of [BH14a].

6.6. Related Work

The connection between safety properties and execution monitors is elaborated in [Sch00]. Information flow security is of a different nature than safety properties. In [Man00a], possibilistic information flow properties are characterized as closure properties on the whole sets of traces of a system. Hence, removing traces in order to enforce a safety property can invalidate such a closure property. This explains the refinement paradox, which was already observed in early works such as [Jac89].

The idea of using composition for the security-preserving enforcement of safety properties also appears in [McL96, Section 3.2] for the framework of McLean's selective interleaving functions. However, this classic security framework does not

consider declassification. We elaborate and lift the idea into the context of BD Security [KLP14], extending our earlier work [BH14a], which in turn relies on the MAKS framework and its compositionality results [Man02].

In the context of MAKS, a paper with a goal very similar to ours is [Man01b]. The approach is different, however. It requires a proof of security of the target system via unwinding, and then modifies the safety property to be enforced by removing or adding traces so that the unwinding conditions are preserved. It works with arbitrary safety properties, but the result can be hard to predict, as it depends heavily on the unwinding relation that is used. We see this approach as complementary to ours.[5] It can be used if compatibility results as we presented them above are not available for the safety property in question, or the side conditions for compositionality are too strong.

There are approaches for security-preserving process refinement (i.e. reducing the set of possible traces) also for other notions of information flow security. [San08] considers confidentiality-preserving refinement for probabilistic information flow. [SS06] builds upon the MAKS framework, but modifies the notions of system specification and security predicates to make the distinction between underspecification and unpredictability explicit. [BFPR03] uses a similar approach to [Man01b], but in the context of a process algebra and bisimulation-based notions of security. Which of the available approaches is best suited for a concrete application depends on the framework used for specifying the system and on the precise security requirements at hand.

6.7. Conclusion

In this chapter, we have focused on the compatibility of possibilistic information flow security and safety properties. We have described how existing compositionality results for information flow predicates can be used to derive sufficient conditions for compatibility with a given safety property. We found this approach to be useful in our case study of verifying the specification of a distributed workflow management system [BH14a].

While Theorem 6.4 applies to arbitrary safety properties, results like our Theorem 6.6 have to be derived for each safety property of interest individually. However, it is worth pointing out that the monitor guarantee for separation of duty is parametric in the event sets and can therefore be instantiated for arbitrary systems where users participate in distinct activities in the presence of separation of duty constraints. Similarly, ordered delivery can be applied to any system with asynchronous message passing. This demonstrates that compositional reasoning can be used to derive monitor guarantees that are applicable to whole classes of common safety properties.

In this chapter, we have considered systems and properties on a high level of

[5] It targets the MAKS framework, but we don't see a reason why it should not be possible to translate it to other security frameworks with an unwinding proof technique, including BD Security.

abstraction, and trace refinement does not change the level of abstraction of a specification. In order to move to a more concrete level of implementation detail, other notions such as action refinement are necessary. Investigating the preservation of security w.r.t. these notions of refinement is an interesting direction for future work.

Chapter 7.

Conclusion

The main theoretical contribution of this thesis is the development of a compositionality result for a highly expressive framework for possibilistic information flow security. Moreover, we discussed different use cases of compositional verification of secure information flow, namely the modular and stepwise development of secure workflow systems; the security-preserving enforcement of safety properties; and a concrete case study of a distributed social media platform. For the latter, we used a compositionality result that leverages a set of simplifying assumptions to ease verification, developed in joint work with the co-authors of [BPPR17].

Our work is based on the framework of Bounded Deducibility (BD) Security [KLP14]. It is highly expressive in terms of the information flow policies it supports, as we have seen with the dynamic declassification policies in our distributed social media case study. Moreover, BD Security can capture many security properties expressible in existing frameworks, e.g. some of the basic security predicates (BSPs) of the MAKS framework [Man00a], or a property similar to PSNI [RS14], as presented in Chapter 2. However, the focus of the original formulation of BD Security was on declassification, and it is not well-equipped to express side conditions for compositionality, in particular regarding the scheduling of secrets and observations. The strengthened variants of BD Security that we define in Section 3.3 are designed for precisely this purpose. They are modeled after the backwards-strict and forward-correctible BSPs of the MAKS framework, and our notion of mutual acceptance of neutral events corresponds to the side conditions of the compositionality result of the MAKS framework [Man02]. Our other conditions serve to support the added expressivity of BD Security: the condition of mutual acceptance of secrets and observations helps with handling the flexibility that views provide to define application-specific attacker models, while the condition of flexible scheduling of secrets and observations helps with handling the flexibility that declassification bounds provide to define security policies. Hence, our framework combines the strengths of BD Security (expressivity) and MAKS (compositionality).

In Chapter 5, we have used compositional reasoning for the stepwise development of secure workflow systems. We discussed a notion of refinement where abstract tasks in a workflow are implemented using a sub-workflow. Formulating these refinements as compositions of sub-workflow and parent workflow allowed us to use our compositionality results to derive security properties. While we focused on workflow systems as our example application domain, the general approach should be

applicable to other domains as well.

In Chapter 6, we showed how compositional reasoning can also be used to preserve BD Security when enforcing safety properties. With this approach, where it is applicable, we obtain a refinement of the system in question, i.e. a system with a subset of the traces of the original system, such that it satisfies both a BD Security property and the safety property. In general, BD Security is not preserved under this notion of refinement, since it is a hyperliveness property, just like other notions of possibilistic information flow security. Using our compositionality result, we have identified sufficient conditions under which such a refinement preserves BD Security. There are also other notions of security formulated as *hypersafety* properties, which are generally preserved under refinement. These include Goguen and Meseguer's original definition of noninterference as well as language-based notions of secure information flow. However, security properties formulated as hypersafety severely restrict the amount nondeterminism that a system may have if it is to be considered secure. For example, observational determinism [ZM03] allows non-observable nondeterminism, but requires the system to appear deterministic to an observer. However, where they are applicable, security definitions in terms of hypersafety often have the advantage that they come with efficient verification techniques. For example, there have been promising advances recently in *model-checking* for temporal logics for hyperproperties [CFK$^+$14]. These techniques also support hyperliveness in principle, but are more efficient for hypersafety security notions due to the lack of quantifier alternations in the logical formulas of typical definitions of the latter. In language-based settings, security type systems or program analysis techniques can often be used to verify security properties automatically. It would be interesting to try to connect some of those security notions to BD Security, as has been done before in the context of the MAKS framework for a language-based security notion [MS03]. Having such a connection for BD Security would allow us to benefit from automatic verification techniques locally in the components of a system, and then use our compositionality result to obtain a BD Security property of the overall system, even if the local security notion does not have a suitable compositionality result.

An interesting direction for future work is the extension of the BD Security framework with a *policy language*. In this thesis, we have used generic logical formulas to specify declassification bounds, but this can become hard to read and understand, especially upon composition. A more tailored language could simplify both the specification and the verification of information flow policies. It is one of the strengths of the MAKS framework [Man00a], for example, that it defines a set of Basic Security Predicates (BSPs) that can be used as simple (albeit rather technical) building blocks for security policies. Our examples suggest that policy templates for BD Security such as "Declassify at most the last version of document x", policy combinators such as "While Q holds, enforce policy P_1, otherwise policy P_2", and a set of results how these building blocks behave under composition, would simplify the specification and verification of policies significantly.

Another interesting direction for future work is to investigate composition and declassification in a *probabilistic* setting. BD Security, like other possibilistic no-

tions of security, does not take into account probabilities: A system is considered secure as long as a sufficient variance of the confidential information is possible from the perspective of the attacker. If, however, an attacker can deduce from their observations that a certain version of the secret has a very high probability, say, an a-posteriori probability of 99%, then this is not considered as an information leak by BD Security, even though it may be of high practical relevance. An interesting research question is whether the compositionality results presented in this thesis can be adapted for probabilistic notions of security. It seems likely that similar challenges and concepts will play a role there, for example the general principle that the local security guarantees of components need to be aligned such that environment assumptions of one component are satisfied by the other component(s).

We believe that a steady progress in the theory of formal security notions as well as in practical tool support, e.g. theorem provers, will enable increased adoption of formal security verification in practice. We hope that our work on compositionality and refinement will contribute to scaling up the verification of complex information flow properties to realistic systems.

Appendix A.

Workflow Specifications

In this appendix, we give a formal specification of the behavior of our activities using PP-statements. In this formalism, the transition relation of a state-event system is specified by listing pre- and post-conditions on the state for each event (see Section 2.1 of [HMSS07] for a formal semantics).

We specify the behavior of our activities in two parts. The PP-statements in Figure A.1 specify the *generic* part of the behavior of activities, i.e. the communication with other activities in order to exchange data items and trigger sequence flows. For this purpose, it maintains program variables $MQueue$ (which data items still have to be sent), $AQueue$ (which data items still have to be acknowledged), $SQueue$ (which triggers still have to be sent), $Triggers$ (whether and from where a trigger has been received), and $User$ (to which user this activity is assigned). The program counters 0, 3 and 4 correspond to the phases of waiting for inputs and triggers, sending outputs, and sending triggers, respectively.

When the program counter reaches 1, an *activity-specific* transition relation takes over in order to perform the actual activity. For our simple example workflows, we only need three kinds of activities:

- user input/output activities, allowing users (with the right clearance) to read and write data items, represented using $Outval$ and $Setval$ events, respectively,
- computation activities, executing a given function f that maps the memory state to a result state, and
- gateway activities, evaluating a condition modeled by a function $Cond$ that takes a memory state and returns two sets of activities: a set of predecessor activities from which triggers must have been received before the workflow may continue, and a set of successor activities to which triggers are sent in order to continue the workflow.

For example, we use the computation activity to model the "Prepare statement" activity in the example workflow W_2, where the function f extracts the equivalence class of the medical report and writes it into the variable containing the medical statement. The gateway activity is used, for example, to model the control flow in the workflow W_3. Depending on the result received from the examination activity and stored in the memory of the gateway, the $Cond$ function returns the set of

$Recv_a(b, Data(i, v))$;
affects: Mem, $AQueue$
Pre: $pc = 0$, $(b, i, a) \in MF$, $(b, i) \notin AQueue$
Post: $Mem'(i) = v$, $AQueue' = AQueue \cup \{(b, i)\}$

$Send_a(b, AckData(i))$;
affects: $AQueue$
Pre: $pc = 0$, $(b, i) \in AQueue$
Post: $AQueue' = AQueue \setminus (b, i)$

$Recv_a(b, Trigger)$;
affects: $Triggers$
Pre: $\top$
Post: $Triggers' = Triggers \cup \{b\}$

τ_a^{Active}; **affects:** pc
Pre: $pc = 0$, $Triggers \neq \emptyset$, $AQueue = \emptyset$
Post: $pc' = 1$

$\tau_a^{SendData}$; **affects:** pc, $MQueue$
Pre: $pc = 2$
Post: $pc' = 3$, $MQueue' = \{(b, i) \mid (a, i, b) \in MF \wedge Mem(i) \neq \bot\}$

$Send_a(b, Data(i, v))$;
affects: $MQueue$, $AQueue$
Pre: $pc = 3$, $(b, i) \in MQueue$
Post: $MQueue' = MQueue \setminus \{(b, i)\}$, $AQueue' = AQueue \cup \{(b, i)\}$

$Recv_a(b, AckData(i))$;
affects: $AQueue$
Pre: $pc = 3$, $(b, i) \in AQueue$
Post: $AQueue' = AQueue \setminus \{(b, i)\}$

$\tau_a^{AckTimeout}$; **affects:** $AQueue$
Pre: $pc = 3$
Post: $AQueue' = \emptyset$

$\tau_a^{SendTriggers}$; **affects:** pc, $SQueue$
Pre: $pc = 3$, $MQueue = \emptyset$, $AQueue = \emptyset$
Post: $pc' = 4$, $SQueue' = \{b \mid (a, b) \in SF\}$

$Send_a(b, Trigger)$; **affects:** $SQueue$
Pre: $pc = 4$, $b \in SQueue$
Post: $SQueue' = SQueue \setminus \{b\}$

Figure A.1.: PP-statements of generic transition relation T_a^{gen}

successor activities (or predecessor activities, in the case of the join gateway complementing the split gateway), possibly including further examinations. For activities involving user participation, we use a Bell-LaPadula style access control. Our specification of user activities only allows users to participate whose clearance matches the classification of the activity. We require that the latter classifications are chosen to respect the classifications of the data items that are available to the activity, i.e.

- for all input data items i of a user activity a, $dom(i) \rightsquigarrow cl_{\mathcal{A}}(a)$, and
- for all output data items i of a user activity a, $cl_{\mathcal{A}}(a) \rightsquigarrow dom(i)$.

This serves to rule out information leaks where a user with insufficient clearance directly accesses a data item with high classification.

The above types of activities are specified in Figures A.2, A.3, and A.4, respectively. Moreover, for workflow refinement, we use a proxy that replaces the refined activity in the parent workflow and is responsible for the communication with the new subworkflow. Its behavior is specified in Figure A.5. The overall transition relation of an activity is the union of the generic transition relation T_a^{gen} of the wrapper and the activity-specific transition relation.

$Start_a(u)$; **affects:** $User$
Pre: $pc = 1$, $User = \bot$, $cl_U(u) = cl_{\mathcal{A}}(a)$
Post: $User' = u$

$Setval_a(u, i, v)$; **affects:** Mem
Pre: $pc = 1$, $User = u$
Post: $Mem'(i) = v$

$Outval_a(u, i, v)$; **affects:**
Pre: $pc = 1$, $User = u$, $Mem(i) = v$

$End_a(u)$; **affects:** pc
Pre: $pc = 1, User = u$
Post: $pc' = 2$

Figure A.2.: PP-statements of transition relation T_a^{user} for user activities

---

τ; **affects:** Mem, pc
Pre: $pc = 1$
Post: $Mem' = f(Mem)$, $pc' = 2$

Figure A.3.: PP-statement of transition relation $T_a^{comp,f}$ for computation activities

---

τ; **affects:** $SQueue$, pc
Pre: $pc = 1$, $Cond(Mem) = (Triggers, TO)$, $\forall b \in TO.\ (a, b) \in SF$
Post: $SQueue = TO$, $pc' = 4$

Figure A.4.: PP-statement of transition relation $T_a^{gw,Cond}$ for gateways

After completion of the activity has been signaled by setting the program counter to 2, the generic transition relation takes control again and starts sending output data items to the designated receivers. It makes sure that they have been received by waiting for acknowledgments, and afterwards proceeds by sending triggers to the successor activities in the workflow. An exception to this rule is if a receiver fails to send an acknowledgment; in this case the $\tau_a^{AckTimeout}$ event can be used to signal a timeout and proceed with the workflow. This is important for security, because otherwise a confidential activity could block the progress of the workflow by refusing to acknowledge a data item. An alternative approach is to enforce ordered delivery of messages, which we discuss in Section 6.5. This way, it is guaranteed that control flow triggers are received *after* data items, so there is no need to wait for an acknowledgment of data items before sending triggers.

τ_a^{Init}; **affects:** pc, $SendInputs$
Pre: $pc = 1$
Post: $pc = 5$, $SendInputs = \{(b, d) \mid (Env, d, b) \in MF(W')\}$

$Send_a(b, Data(d, v))$;
affects: $SendInputs$
Pre: $pc = 5$, $(b, d) \in SendInputs$, $Mem(d) = v$
Post: $SendInputs' = SendInputs \setminus \{(b, d)\}$

$Send_a(b, Trigger)$; **affects:** pc
Pre: $pc = 5$, $(Env, b) \in SF(W')$, $SendInputs = \emptyset$
Post: $pc' = 6$

$Recv_a(b, Data(d, v))$; **affects:** Mem
Pre: $pc = 6$, $(b, d, Env) \in MF(W')$
Post: $Mem'(d) = v$

$Recv_a(b, Trigger)$; **affects:** pc
Pre: $pc = 6$, $(b, Env) \in SF(W')$
Post: $pc' = 2$

Figure A.5.: PP-statements of transition relation $T^{Proxy(a,W')}$

Appendix B.

Unwinding of BD Security

We sketch the use of the unwinding proof method of [KLP14, Section 5] for the Example in Section 2.2.5. The states of the example system can be defined as a triple of a natural number representing the step in the workflow, a candidate name, and a medical report. The start state is $(0, \bot, \bot)$. The transition relation is defined inductively via the rules:

$$
\begin{aligned}
&(0, \bot, \bot) \overset{examine?c}{\rightarrow} (1, c, \bot) \\
&(1, c, x) \overset{report?x'}{\rightarrow} (1, c, x') \\
&(1, c, x) \overset{finish?\mathrm{True}}{\rightarrow} (2, c, x) && \text{if } x \neq \bot \\
&(2, c, x) \overset{stmt!(c', d')}{\rightarrow} (3, c, x) && \text{if } decision(x) = d' \wedge c = c'
\end{aligned}
$$

Our unwinding relation Δ consists of the quadruples $(\sigma, sl, \sigma', sl')$ such that

- $(sl, sl') \in B_{Med}$ or $sl = sl' = \langle\rangle$ and
- for some pc, c, x, x', $\sigma = (pc, c, x)$, $\sigma' = (pc, c, x')$, and, if $sl = sl' = \langle\rangle$, then $x \approx_{Med} x'$.

We use the following unwinding strategy: If $pc = 1$ and sl is not empty, we perform *ignore* or *independent action* steps, respectively, in order to consume sl and sl' so that both contain only one remaining report. In all other cases, we perform a *match* step, producing the same event from both states. It is easy to show that each of the above steps is possible in the states σ and σ' at hand, and that the unwinding conditions of [KLP14] are satisfied. Moreover, regarding the initial state, $(\sigma_0, sl, \sigma_0, sl') \in \Delta$ holds for all $(sl, sl') \in B_{Med}$.

The theoretical core of the unwinding technique is a lemma saying that, if the unwinding conditions are satisfied, then for any $(\sigma, sl, \sigma', sl') \in \Delta$ and $\sigma \overset{t}{\Rightarrow}$ with $S_{\mathcal{V}}(t) = sl$, there is a t' with $\sigma' \overset{t'}{\Rightarrow}$, $S_{\mathcal{V}}(t') = sl'$, and $O_{\mathcal{V}}(t') = O_{\mathcal{V}}(t)$. The soundness of unwinding for BD Security follows by instantiating the lemma for $(\sigma_0, sl, \sigma_0, sl') \in \Delta$ for a given $(sl, sl') \in B$.

It is straightforward to adapt this technique for the unwinding of the side conditions we presented in Chapter 3. We modify the unwinding lemma only by changing the initial condition to incorporate the corresponding requirements listed

Table B.1.: Unwinding of side conditions

Condition[a,b]	$P'(t, sl')$ [c]	$Q'(t, t')$ [c]
$BSBD$	$t = \langle\rangle$	$t' = \langle\rangle$
$EI[X]$	$t = \langle x \rangle$	$t' = \langle x' \rangle$
$LR[R]$	$s\, R\, s' \wedge t = \langle s \rangle \wedge getSec(s') \leq sl'$	$t' = \langle s' \rangle$
$IB[X, Y, Z]$	$t = \langle z \rangle \wedge S_{\mathcal{V}}(\langle x, z \rangle) \leq sl'$	$t' = x' \cdot ys' \cdot z' \wedge z' \approx_{\mathcal{V}} \mathrm{hd}(t)$
$DB[X, Y, Z]$	$t = x \cdot ns \cdot z \wedge S_{\mathcal{V}}(\langle z \rangle) \leq sl'$	$t' = ys' \cdot z' \wedge z' \approx_{\mathcal{V}} \mathrm{hd}(t \restriction Z)$

[a] where we omit the parameters $[\ldots, \mathcal{V}, B]$ for brevity
[b] where $\mathcal{V} = (Ev^{obs}, getObs, Ev^{sec}, getSec)$, $X \subseteq Ev^{sec}$, $Y \subseteq N^{\mathcal{V}}_{LTS}$, and $Z \subseteq Ev^{obs} \cup Ev^{sec}$
[c] Free variables in each cell are existentially quantified, such that s, s' range over Ev^{sec}, ns over $(N^{\mathcal{V}}_{LTS})^*$, x, x', x'' over $X \cap Ev^{sec}$, z, z' over $Z \cap (Ev^{obs} \cup Ev^{sec})$, and ys, ys' over $(N^{\mathcal{V}}_{LTS} \setminus Y)^*$.

in Table 3.1. For example, when unwinding $EI[X, \mathcal{V}, B]$, we require that for all $\beta, \alpha \in Ev^*$, $\sigma \in St$, $x \in X \cap Ev^{sec}$, and $sl' \in \mathsf{Sec}^*$ with $\sigma_0 \stackrel{\beta}{\Rightarrow} \sigma \stackrel{\alpha}{\Rightarrow}$ and $(S(\beta) \cdot S(\alpha), S(\beta) \cdot getSec(x) \cdot sl') \in B$, there is some $e \in X \cap Ev^{sec}$ such that $\sigma \stackrel{e}{\rightarrow} \sigma'$, $getSec(e) = getSec(x)$, and $(\sigma, S(\alpha), \sigma', sl') \in \Delta$. In addition, the unwinding conditions for Δ have to hold as above. For a given $\beta \cdot \alpha \in [\![LTS]\!]$, we then invoke the above lemma to obtain a suitable α'. Unwinding conditions for the other properties we presented in Chapter 3 are constructed analogously. Table B.1 summarizes them, where P' captures the conditions on prefixes t of α under which the property applies, and Q' captures the requirements on suitable prefixes t' of α'.

Theorem B.1. *Let (P, Q) be one of the side conditions of Table 3.1 for a view $\mathcal{V}$ and a bound B, and let (P', Q') be the corresponding unwinding conditions given in Table B.1. If there is an unwinding relation $\Delta \subseteq St \times \mathsf{Sec}^* \times St \times \mathsf{Sec}^*$ such that*

- *for all β, t, sl, and sl' such that $\sigma_0 \stackrel{\beta}{\Rightarrow} \sigma \stackrel{t}{\Rightarrow} \sigma'$, $(S(\beta) \cdot S(t) \cdot sl, S(\beta) \cdot sl') \in B$, and $P'(t, sl')$, there are t', σ'' and sl'' such that $\sigma \stackrel{t'}{\Rightarrow} \sigma''$, $Q'(t, t')$, $sl' = S(t') \cdot sl''$, $O(t') = O(t)$, and $(\sigma', sl, \sigma'', sl'') \in \Delta$, and*
- $\mathsf{unwind}(\Delta)$ *holds as defined in [KLP14, Section 5],*

then LTS satisfies the BD Security property with the side condition (P, Q) w.r.t. $\mathcal{V}$ and B.

Proof idea. The conditions in Table B.1 directly mirror the conditions on the initial parts of the traces α and α' given in Table 3.1. The possibility of producing the remaining secrets sl and sl'' from the states σ and σ'', respectively, while producing the same observations, follows from $(\sigma', sl, \sigma'', sl'') \in \Delta$ together with $\mathsf{unwind}(\Delta)$ and the original unwinding lemma of [KLP14, Section 5]. □

Appendix C.

Proofs

C.1. Background

Proof sketch of Lemma 2.1. We show that for all $t \in Ev^*$,

$$\forall \sigma_1, \sigma_2.\ \left((\sigma_1, \sigma_2) \overset{t}{\Rightarrow}\right) \longleftrightarrow \left(\sigma_1 \overset{t\restriction 1}{\Longrightarrow}_1\right) \wedge \left(\sigma_2 \overset{t\restriction 2}{\Longrightarrow}_2\right)$$

via induction on t. The base case $t = \langle\rangle$ holds trivially. In the inductive step $t = e \cdot t'$, we perform a case distinction on e: due to the construction of Ev in Definition 2.4, either $e = (1, e_1)$ with $e_1 \in Ev_1 \setminus If_1$, or $e = (2, e_2)$ with $e_2 \in Ev_2 \setminus If_2$, or $e = (e_1, e_2)$ with $e_i \in If_i$ and $(e_1 \parallel e_2 \vee e_2 \parallel e_1)$. We apply the corresponding rules of Figure 2.4 and the definition of trace projection ($\restriction$), and use the induction hypothesis on the successor state after e.

The lemma then follows by unfolding $[\![LTS]\!]$ and $[\![LTS_i]\!]$ and plugging in the initial states. □

Lemma 2.2 is proved analogously.

C.2. Compositionality

Proof of Theorem 3.1. Let $t \in [\![Sys]\!]$ and $(S(t), s') \in B$. Since Sys is a composition of Sys_1 and Sys_2 via $\oplus$, we obtain $t_1 \in [\![Sys_1]\!]$ and $t_2 \in [\![Sys_2]\!]$ such that $t \in t_1 \oplus t_2$. With the definition of $\oplus_O$ and $\oplus_S$, we get

$$O(t) \in O_1(t_1) \oplus_O O_2(t_2) \tag{C.1}$$

$$S(t) \in S_1(t_1) \oplus_S S_2(t_2) \tag{C.2}$$

From $B \subseteq B_1 \oplus_B^C B_2$ and the definition of $B_1 \oplus_B^C B_2$, we obtain s_1' and s_2' such that

$$s' \in s_1' \oplus_S s_2' \tag{C.3}$$

and $(S_i(t_i), s_i') \in B_i$ holds. Since each Sys_i satisfies BD Security w.r.t. O_i, S_i, and B_i, we obtain $t_i' \in [\![Sys_i]\!]$ such that $S_i(t_i') = s_i'$ and $O_i(t_i') = O_i(t_i)$. Substituting this into (C.1) and (C.3) gives $O(t) \in O_1(t_1') \oplus_O O_2(t_2')$ and $s' \in S_1(t_1') \oplus_S S_2(t_2')$.

Moreover, from $(S(t), s') \in B$ and the definition of $B_1 \oplus_B^C B_2 \supseteq B$, we get $S(t) \preceq_C s'$. Since $(O(t), S(t)) \in C$ due to the fact that C is an observation-secret compatibility relation between O and S, this implies $(O(t), s') \in C$.

With the well-behavedness of Sys, we obtain $t' \in [\![Sys]\!]$ such that $S(t') = s'$ and $O(t') = O(t)$. □

Proof of Lemma 3.2. The fact that, if e_i is not secret, both e_j and e_i must be observable and non-secret follows immediately from view composability (Definition 3.7).

Let $e'_j \in Out_j$ with $e'_j \approx_{\mathcal{V}_j} e_j$ be given. From the fourth condition of Definition 3.15 we obtain an $e'_i \in In_i$ with $e'_j \parallel e'_i$. We have $e_i \not\bowtie_{i,j} e'_i$, since $e_i \bowtie_{i,j} e'_i$ would contradict the assumption that $e_i \notin Ev_i^{sec}$ due to the first condition of Definition 3.15. By instantiating the negation of Definition 3.12 for e_i, e'_i, e_j, and e'_j, we obtain $e'_j \parallel e_i$. □

In order to prove the compositionality of BD Security properties of LTSs with and without side conditions, we first prove a lemma analogous to Mantel's Generalized Zipping Lemma.

Lemma C.1. *Let the preconditions of Lemma 3.3 hold. Let*

- $\beta \in \beta_1 \parallel \beta_2$,
- $\beta_1 \cdot \alpha_1 \in [\![LTS_1]\!]$ *and* $\beta_2 \cdot \alpha_2 \in [\![LTS_2]\!]$,
- $ol \in O_1(\alpha_1) \parallel_O O_2(\alpha_2)$ *and* $sl \in S_1(\alpha_1) \parallel_S S_2(\alpha_2)$,
- $\alpha_1 \restriction \overline{N_2} = \langle\rangle$ *and* $\alpha_2 \restriction \overline{N_1} = \langle\rangle$, *and*
- $(ol, sl) \in C$.

Then there are α', α'_1, *and* α'_2 *such that*

- $\alpha' \in \alpha'_1 \parallel \alpha'_2$,
- $\beta_1 \cdot \alpha'_1 \in [\![LTS_1]\!]$ *and* $\beta_2 \cdot \alpha'_2 \in [\![LTS_2]\!]$, *and*
- $O(\alpha') = ol$ *and* $S(\alpha') = sl$.

Proof. By induction on $|ol| + |sl|$.

In the base case ($|ol| + |sl| = 0$), there are no more observations and secrets to produce, i.e., $ol = sl = \langle\rangle$. We simply choose $\alpha' = \alpha'_1 = \alpha'_2 = \langle\rangle$. Since $\beta_i \cdot \alpha_i \in [\![LTS_i]\!]$ implies $\beta_i \in [\![LTS_i]\!]$, the conclusion follows immediately.

In the inductive step, we first distinguish which of the cases of the third condition of Lemma 3.3 holds.

- Both LTS_1 and LTS_2 support flexible scheduling of secrets and observations for $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively. We make a case distinction on the first element(s) of ol and sl:
  1. $ol = o \cdot ol'$ with $o \notin \mathsf{Obs}^{\mathsf{Sec}}$. We proceed with a case distinction on o:

a) $o = (1, o_1)$. Hence, $O_1(\alpha_1) = o_1 \cdot ol'_1$ for some ol'_1. We split α_1 accordingly into $\beta'_1 \cdot e_1 \cdot \alpha'_1$ such that $O_1(\beta'_1) = \langle\rangle$, $e_1 \in Ev_1^{obs} \setminus Ev_1^{sec}$, $e_1 \notin If_1$, $getObs(e_1) = o_1$, $O_1(\alpha'_1) = ol'_1$. Any secret events in β'_1 must be non-observable due to $O_1(\beta'_1) = \langle\rangle$ and not match a neutral event of LTS_2 due to $\alpha_1 \upharpoonright \overline{N_2} = \langle\rangle$. Since LTS_1 supports the deletion of $(Ev_1^{sec} \setminus Ev_1^{obs})$ before $(Ev_1^{obs} \setminus Ev_1^{sec})$ due to the flexible scheduling of secrets and observations, we can move the secret events in β'_1 after the event e_1 via an inductive argument. We obtain $\beta''_1 \cdot e'_1 \cdot \alpha''_1$ with $O_1(\beta''_1) = S_1(\beta''_1) = \langle\rangle$, $e'_1 \approx_{\mathcal{V}_1} e_1$, $O_1(\alpha''_1) = ol'_1$, $S_1(\alpha''_1) = S_1(\alpha_1)$, and $\beta_1 \cdot \beta''_1 \cdot e'_1 \cdot \alpha''_1 \in [\![LTS_1]\!]$. For each neutral interface event in β''_1 (which must be an output event by Definition 3.7), we inductively insert a matching input event into α_2, using the fact that LTS_2 accepts the neutral events of $\mathcal{V}_1$ via $\|$, obtaining $n_2 \in (\overline{N_1})^*$, α''_2, and a β'' such that $\beta'' \in \beta''_1 \parallel n_2$, $O(\beta'') = S(\beta'') = \langle\rangle$, $O_2(\alpha''_2) = O_2(\alpha_2)$, $S_2(\alpha''_2) = S_2(\alpha_2)$, and $\beta_2 \cdot n_2 \cdot \alpha''_2 \in [\![LTS_2]\!]$. We have now "closed the zipper" up to e'_1, i.e., $\beta \cdot \beta'' \cdot (1, e'_1) \in [\![LTS]\!]$. We apply the induction hypothesis to merge α''_1 and α''_2 to a composed trace producing the remaining observations ol' and secrets sl.

b) $o = (2, o_2)$. Symmetric to the previous case.

c) $o = o_1 *_O o_2$ such that there are β'_i, e_i, and α'_i with $getObs_i(e_i) = o_i$, $O_i(\beta'_i) = \langle\rangle$, and $\alpha_i = \beta'_i \cdot e_i \cdot \alpha'_i$. Let e_1 be an output event of LTS_1 and e_2 be an input event of LTS_2 (the proof for the other direction is symmetric). Since $o \notin \mathsf{Obs}^{\mathsf{Sec}}$ and the views are well-defined, we have $e_i \in Ev_i^{obs} \setminus Ev_i^{sec}$. We consider different cases regarding the distribution of neutral events.

   - $\overline{N_2} \cap Ev_1 = \emptyset$. We proceed analogously to case a) up to the point where we match the neutral output events in β''_1 by inserting corresponding input events n_2 into the trace of LTS_2. The difference to a) is that we now also have to take into account e_2, since $\alpha_2 = \beta'_2 \cdot e_2 \cdot \alpha'_2$. We again use the fact that LTS_2 accepts the neutral events of $\mathcal{V}_1$ via $\|$ to obtain $\beta_2 \cdot n_2 \cdot \beta''_2 \cdot e'_2 \cdot \alpha''_2 \in [\![LTS_2]\!]$ and β'' such that $\beta'' \in \beta''_1 \parallel n_2$, $O(\beta'') = S(\beta'') = \langle\rangle$, $O(\beta''_2) = \langle\rangle$, $e'_2 \approx_{\mathcal{V}_2} e_2$, $O_2(\alpha''_2) = O_2(\alpha'_2)$, $S_2(\beta''_2 \cdot e'_2 \cdot \alpha''_2) = S_2(\alpha_2)$. Now β''_2 might contain secret events, but since $e'_2 \in Ev_2^{obs} \setminus Ev_2^{sec}$ holds and LTS_2 supports flexible scheduling of secrets and observations, we can move those secret events after e'_2 and obtain β'''_2, e''_2, and α'''_2 with $\beta_2 \cdot n_2 \cdot \beta'''_2 \cdot e''_2 \cdot \alpha'''_2 \in [\![LTS_2]\!]$, $e''_2 \approx_{\mathcal{V}_2} e_2$, $O_2(\alpha'''_2) = O_2(\alpha'_2)$, $S_2(\alpha'''_2) = S_2(\alpha'_2)$, and $O(\beta'''_2) = S(\beta'''_2) = \langle\rangle$. The latter means $\beta'''_2 \in (N_2)^*$, so it does not contain any communication with LTS_1 due to the assumption $\overline{N_2} \cap Ev_1 = \emptyset$ in this case. Hence, we can synchronize the traces of the two components up to right before e'_1 and e''_2, respectively. However, we also have $e'_1 \parallel e''_2$: Note that there are *some* synchronizing events $e_1^0 \parallel e_2^0$ with $e_1^0 \approx_{\mathcal{V}_1} e'_1$ and

$e_2^0 \approx_{\mathcal{V}_2} e_2''$, since we know that the observations of e_1' and e_2'' do synchronize (to o). This implies $e_1^0 \parallel e_2''$ due to the second condition of view composability (Definition 3.7). With $e_1^0 \approx_{\mathcal{V}_1} e_1'$ and $e_2'' \notin Ev_2^{sec}$ we get $e_1' \parallel e_2''$ via Lemma 3.2. Hence, we can merge the traces up to and including (e_1', e_2''). Finally, we use the induction hypothesis to merge the remaining traces α_1'' and α_2'''' to a composed trace producing the remaining observations ol' and secrets sl.

- $\overline{N_1} \cap Ev_2 = \emptyset$. Symmetric to the previous case.
- $\overline{N_2} \cap Ev_1 \neq \emptyset \neq \overline{N_1} \cap Ev_2$. Since LTS_1 and LTS_2 accept each other's neutral events, there are Z_1 and Z_2 that cover their interface. Hence, either $e_1 \in Z_1$ or $e_2 \in Z_2$. Let the first case hold (the other one is symmetric). We proceed like in the case $\overline{N_2} \cap Ev_1 = \emptyset$, up to the point where we are about to synchronize e_1' and e_2''. The difference is that now β_2''' might contain neutral communication events which have to be matched by LTS_1. As per Definition 3.11, LTS_1 silently accepts the neutral events of $\mathcal{V}_2$ before Z_1. Since $e_1 \approx_{\mathcal{V}_1} e_1'$, we also have $e_1' \in Z_1$ due to Definition 3.10. This allows us to insert a sequence n_1 of input events in front of $e_1' \in Z_1$ that match the neutral output events in β_2''', obtaining $\beta_1'' \cdot n_1 \cdot e_1'' \cdot \alpha_1'''$ and $\beta''' \in n_1 \parallel \beta_2'''$. Since n_1 does not contain output events, there is no need to add further events to the trace of LTS_2. Hence, we can synchronize β_1'' with n_2 and n_1 with β_2''', yielding the composed trace $\beta \cdot \beta'' \cdot \beta''' \in [\![LTS]\!]$. Now we can again close the zipper up to (e_1'', e_2'''), applying the induction hypothesis to merge the remaining traces α_1''' and α_2'''.

2. $sl = s \cdot sl'$ with $s \notin \mathsf{Sec}^{\mathsf{Obs}}$. We proceed with a case distinction on s.

   a) $s = (1, s_1)$. The proof proceeds analogously to case 1a): We obtain $S_1(\alpha_1) = s_1 \cdot sl_1'$ for some sl_1', and we split α_1 into $\beta_1' \cdot e_1 \cdot \alpha_1'$ with $S_1(\beta_1') = \langle\rangle$, $e_1 \in Ev_1^{sec} \setminus Ev_1^{obs}$, $e_1 \notin If_1$, $getSec(e_1) = s_1$, $S_1(\alpha_1') = sl_1'$. The main difference to case 1a) is that we now use the insertion of $(Ev_1^{sec} \setminus Ev_1^{obs})$ before $(Ev_1^{obs} \setminus Ev_1^{sec})$ to move the secret event e_1 *backwards* in front of any observable events in β_1', obtaining $\beta_1'' \cdot e_1' \cdot \alpha_1''$ with $O_1(\beta_1'') = S_1(\beta_1'') = \langle\rangle$, $e_1' \approx_{\mathcal{V}_1} e_1$, $O_1(\alpha_1'') = O_1(\alpha_1)$, $S_1(\alpha_1'') = sl_1'$, and $\beta_1 \cdot \beta_1'' \cdot e_1' \cdot \alpha_1'' \in [\![LTS_1]\!]$. As in case 1a), we proceed by inserting a sequence of events n_2 into the trace of LTS_2 to synchronize with the neutral interface events in β_1'', merge the traces up to e_1', and apply the induction hypothesis to merge α_1'' and α_2'' to a composed trace producing the remaining secrets sl' and observations ol.

   b) $s = (2, s_2)$. Symmetric to the previous case.

   c) $s = s_1 *_S s_2$. The proof proceeds analogously to case 1c). Again, one

difference is that we use insertion instead of deletion of $(Ev_i^{sec} \setminus Ev_i^{obs})$ before $(Ev_i^{obs} \setminus Ev_i^{sec})$ to move the secret events backwards, in front of the observable events. The synchronization of neutral events using the fact that the LTS_i (silently) accept each other's neutral events proceeds like above. However, in this case it is not guaranteed that the two communication events e_1'' and e_2'' synchronize: while the secrets of input events contain all the information relevant for synchronization due to the requirements of view composability (Definition 3.7), the secrets of output events may abstract away some of that information. Let us assume that e_1'' is an output event and e_2'' is an input event (the proof for the other case proceeds symmetrically). The fact that $s = s_1 *_S s_2$ only guarantees that there are *some* equivalent and synchronizing events $e_1^0 \parallel e_2^0$ with $e_i^0 \approx_{\mathcal{V}_i} e_i''$. In case $e_1'' \parallel e_2''$ does not happen to hold already, we first obtain another event e_2''' with $e_1'' \parallel e_2'''$ via the fact that $\mathcal{V}_1$ and $\mathcal{V}_2$ have complementary secrets and observations (in particular, via the fourth condition of Definition 3.15). With $e_1'' \nparallel e_2''$, we have $e_2'' \bowtie_{2,1} e_2'''$ (cf. Definition 3.12). Since LTS_2 supports local replacement via $\bowtie_{2,1}$ (cf. Definition 3.17), we can replace e_2'' by e_2''', also obtaining α_2'''' (following e_2''') with the same secrets and observations as α_2'''. We can now synchronize the events e_1'' and e_2'''. Note that the composite event (e_1'', e_2''') still produces the desired secret s due to the third condition of Definition 3.15 (or the second condition in the symmetric case $e_2 \in Out_2$). Moreover, it does not produce an observation, since the two events are still not observable: $e_1'' \notin Ev_1^{obs}$ due to $e_1 \notin Ev_1^{obs}$ and $e_1'' \approx_{\mathcal{V}_1} e_1$ together with well-definedness of $\mathcal{V}_1$ (Definition 3.20), and $e_1'' \notin Ev_1^{obs}$ also implies $e_2''' \notin Ev_2^{obs}$ due to $e_1'' \parallel e_2'''$ and composability of $\mathcal{V}_1$ and $\mathcal{V}_2$ (Definition 3.7). Finally, we apply the induction hypothesis as usual to merge the remaining traces.

3. $ol = o \cdot ol'$ with $o \in \mathsf{Obs}^{\mathsf{Sec}}$ and $sl = s \cdot sl'$ with $s \in \mathsf{Sec}^{\mathsf{Obs}}$. We proceed with a case distinction on o.

   a) $o = (1, o_1)$ with $o_1 \in \mathsf{Obs}_1^{\mathsf{Sec}}$. Since $(ol, sl) \in C$, we get $s = (1, s_1)$ with $s_1 \in \mathsf{Sec}_1^{\mathsf{Obs}}$. Hence, $O_1(\alpha_1) = o_1 \cdot ol_1'$ and $S_1(\alpha_1) = s_1 \cdot sl_1'$ for some ol_1' and sl_1'. Both o_1 and s_1 must have been produced by an event in $Ev_1^{obs} \cap Ev_1^{sec}$, because observations in $\mathcal{V}_1$ indicate whether they have been produced by a secret event (and vice versa) due to the well-definedness of $\mathcal{V}_1$ (cf. Definition 3.20). Hence, they must have been produced in α_1 by the *same* event $e_1 \in Ev_1^{obs} \cap Ev_1^{sec}$, and we split α_1 into $\beta_1' \cdot e_1 \cdot \alpha_1'$ with $O_1(\beta_1') = S_1(\beta_1') = \langle\rangle$, $getObs_1(e_1) = o_1$, $getSec_1(e_1) = s_1$, $O_1(\alpha_1') = ol_1'$ and $S_1(\alpha_1') = sl_1'$. Since β_1' does not contain any observable or secret events, there is no need to move any events, and we directly proceed to synchronize its neutral interface events by inserting n_2 into the trace of LTS_2 as usual using the fact

that LTS_2 accepts the neutral events of LTS_1. Finally, we apply the induction hypothesis to merge α_1' and α_2' to a composed trace producing ol' and sl'.

b) $o = (2, o_2)$ with $o_2 \in \mathsf{Obs}_2^{\mathsf{Sec}}$. Symmetric to the previous case.

c) $o = (o_1, o_2)$ with $o_i \in \mathsf{Obs}_i^{\mathsf{Sec}}$. Analogously to case a), we split the α_i into $\beta_i' \cdot e_i \cdot \alpha_i'$. Again, the β_i' do not contain observable or secret events, so there is no need to move events. Otherwise, we proceed analogous to case 2c), mutually inserting neutral events using the fact that the LTS_i (silently) accept each other's neutral events, synchronizing the e_i' (possibly with local replacement of the input event), and merging the remaining traces using the induction hypothesis.

These three cases form a complete case distinction in the inductive step, since the case where ol and sl are both empty is the base case above, and the remaining cases are not possible: if ol begins with an $o \in \mathsf{Obs}^{\mathsf{Sec}}$, then sl cannot be empty either due to $(ol, sl) \in C$ (and vice versa).

- $Ev_1^{obs} \subseteq Ev_1^{sec} \cap If_1$ or $Ev_2^{obs} \subseteq Ev_2^{sec} \cap If_2$. Let the first condition hold (the proof for the other one is symmetric). We make a case distinction on sl.

  1. $sl = \langle\rangle$. Then ol must be non-empty and consist of observations of LTS_2, because observable events of LTS_1 would also produce secrets. Hence, $ol = (2, o_2) \cdot ol'$ with $o_2 \notin \mathsf{Obs}_2^{\mathsf{Sec}}$. We proceed like in the case 1b) above, but without having to move any secret events.

  2. $sl = s \cdot sl'$. We proceed with a case distinction on s.

     a) $s = (1, s_1)$. We split α_1 into $\beta_1' \cdot e_1 \cdot \alpha_1'$ as usual. Since s_1 is local, e_1 cannot be an observable event due to the above assumption. Moreover, β_1' cannot contain observable events, since those would have produced secrets. We proceed like in the case 2a) above, but without having to move observable events.

     b) $s = (2, s_2)$. We split α_2 into $\beta_2' \cdot e_2 \cdot \alpha_2'$ as usual. If there are no observable events in β_2', we proceed like in case 2b) or 3b) above, depending on whether e_2 itself is observable, but without having to move any observable events. If there are observable events in β_2', then ol must be non-empty and we perform a case distinction on the first element of ol.

        - $ol = (1, o_1) \cdot ol'$. This case is not possible due to the above assumption: LTS_1 does not produce local observations.
        - $ol = (2, o_2) \cdot ol'$. Hence, o_2 must be the first observation in β_2', produced by some event e_2'. Note that it is not preceded by any secret events; the first secret event e_2 comes *after* β_2'. We proceed like in the case 1b) above, but without having to move any secret events; it suffices to synchronize the neutral communication

events in β_2' by inserting matching input events in the trace of LTS_1. We apply the induction hypothesis to obtain a composed trace producing the remaining observations ol' and secrets sl.

- $ol = (o_1, o_2) \cdot ol'$. This case is not possible: Again, o_2 must be the first observation in β_2', produced by some event e_2'. Due to the above assumption, the event producing o_1 must contain secret information. Due to the composability of the views (cf. Definition 3.7), this implies that e_2' must be secret as well, which contradicts the fact that β_2' does not produce secrets.

c) $sl = (s_1, s_2) \cdot sl'$. Again, we split α_2 into $\beta_2' \cdot e_2 \cdot \alpha_2'$. If there are observable events in β_2', then they must be local to LTS_2, and we proceed to directly merge the first one of them and invoke the induction hypothesis like in the previous case. If there are no observable events in β_2', then we proceed like in the case 2c) or 3c) above, depending on whether e_2 is itself observable. The difference to those cases is, again, that we do not need to move any observable events: LTS_1 cannot produce purely observable events (they would be reflected in sl as well), there are none in β_2', and if an observable event gets moved before the secret event after we have inserted neutral events into β_2'', then we again proceed directly to merge the first observable event instead of e_2 before applying the induction hypothesis.

- $Ev_1^{sec} \setminus \overline{N_2} \subseteq Ev_1^{obs} \cap If_1$ or $Ev_2^{sec} \setminus \overline{N_1} \subseteq Ev_2^{obs} \cap If_2$. We proceed like in the previous case, but with the roles of secrets and observations swapped. We start with a case distinction on the *observations*. In each of the cases, the above assumption allows us to conclude that we do not have to move any *secret* events. The synchronization of neutral and communication events proceeds as usual. □

Proof of Lemma 3.3. We first invoke the BD Security properties of the LTS_i to remove the events in $\overline{N_j}$ while preserving the other secrets and observations. This makes use of the fact that the LTS_i accept each other's neutral events, which implies that they satisfy BD Security w.r.t. B_i that are total in $\overline{N_j}$ (from Definition 3.11), and the fact that the $\mathcal{V}_i$ distinguish $\overline{N_j}$ (from the second condition of Definition 3.7). Hence, we can apply BD Security using $(S(t_i), S(t_i) \setminus S(\overline{N_j})) \in B_i$. Afterwards, well-behavedness follows immediately from Lemma C.1 by setting $\beta = \beta_1 = \beta_2 = \langle\rangle$, $\alpha_1 = t_1'$, $\alpha_2 = t_2'$. □

Proof of Theorem 3.4. By instantiating Theorem 3.1. □

Proof of Theorem 3.5. As discussed already in the main text, the preconditions of Theorems 3.4 and 3.7 are satisfied.

Let $\mathcal{V}$ denote the (canonical) composition of $\mathcal{V}_{L_1,M_1,H_1}$ and $\mathcal{V}_{L_2,M_2,H_2}$ w.r.t. $\|$, $*_S$, and $*_O$, where we define the latter analogously to $*_S$. In $\mathcal{V}$, the secret events are:

- local secret events of LTS_1 or LTS_2, which can be input events on H_i channels or input or output events on M_i channels that are *not* shared with the other system;
- composite secret events that are secret for *both* systems, i.e., communication events on shared M channels; note that communication on H_i channels becomes neutral, because output on high channels is neutral.

This corresponds to the classification of M and H channels that we chose for LTS, i.e., $M = M_1 \cup M_2$ and $H = (H_1 \setminus \mathrm{Chans}(If_2)) \cup (H_2 \setminus \mathrm{Chans}(If_1))$. Similarly, the observable events in $\mathcal{V}$ are the events on L and M channels of LTS with $L = L_1 \cup L_2$. Moreover, as noted in the text, the bounds B_{L_i,M_i,H_i} are total, so their canonical composition is total as well (and therefore history-independent and backwards-strict). Finally, the canonical composition of the parameters of the side conditions are at least as strong as needed for the desired side conditions for LTS; for example, the product of Ev_{HI_1} and Ev_{HI_2} contains Ev_{HI}, so LTS still supports eager insertion of Ev_{HI}.

As discussed in the text, the only problem is that the canonical composition of LTS_1 and LTS_2 has a slightly different shape than desired, e.g. events of the form $(1, e_1)$. To remedy this, we use the translation function r for events as defined in the text, as well as

$$r_S(e) = \begin{cases} e_1 & \text{if } e = (1, e_1) \\ e_2 & \text{if } e = (2, e_2) \\ e & \text{otherwise} \end{cases}$$

for secrets and r_O for observations, defined analogously to r_S (note that composed secrets and observations are already handled by the operators $*_S$ and $*_O$ we use here). Since all of these functions are total and bijective, there is a one-to-one correspondence between the traces, secret and observation sequences, and bounds of the original system (the canonical composition of LTS_1 and LTS_2) and LTS. Hence, the security property and the conditions of the former trivially carry over to the latter, and the translated property corresponds to *PSNI* for LTS. □

Proof of Lemma 3.6. Let B_1, B_2, and $B \subseteq B_1 \parallel_B B_2$ be history-independent bounds. Fix $(sl, sl') \in B$. We have to show that $(sl, sl') \in B_1 \parallel_B^{BS} B_2$. For this purpose, fix $\beta, \alpha, \alpha', \beta_1, \beta_2, \alpha_1, \alpha_2$ such that $\beta \in \beta_1 \parallel_S \beta_2$, $\alpha \in \alpha_1 \parallel_S \alpha_2$, $sl = \beta \cdot \alpha$, and $sl' = \beta \cdot \alpha'$. Hence, $(\beta \cdot \alpha, \beta \cdot \alpha') \in B$. Since B is history-independent, we get $(\alpha, \alpha') \in B \subseteq B_1 \parallel_B B_2$. From the definition of $B_1 \parallel_B B_2$, we obtain α'_1, α'_2 such that $\alpha' \in \alpha'_1 \parallel_S \alpha'_2$, $(\alpha_i, \alpha'_i) \in B_i$, and $\alpha \preceq_C \alpha'$. Since the B_i are history-independent, we get $(\beta_i \cdot \alpha_i, \beta_i \cdot \alpha'_i) \in B_i$. Moreover, from $\alpha \preceq_C \alpha'$ and the definition of $\preceq_C$ we get $(\beta \cdot \alpha) \preceq_C (\beta \cdot \alpha')$. □

Proof of Theorem 3.7. All notions of BD Security with side conditions consider a trace $\beta \cdot \alpha \in [\![LTS]\!]$ with $(S(\beta) \cdot S(\alpha), S(\beta) \cdot sl') \in B$. Since LTS is a composition of LTS_1 and LTS_2, we have $\beta \in (\beta \restriction 1) \parallel (\beta \restriction 2)$ and $\alpha \in (\alpha \restriction 1) \parallel (\alpha \restriction 2)$.

Hence, $S(\beta) \in S_1(\beta \restriction 1) \parallel_S S_2(\beta \restriction 2)$ and $S(\alpha) \in S_1(\alpha \restriction 1) \parallel_S S_2(\alpha \restriction 2)$. Since $B \subseteq B_1 \parallel_B^{BS} B_2$, we obtain a decomposition of sl' into sl'_1 and sl'_2 with $sl' \in sl'_1 \parallel_S sl'_2$ and $(S_i(\beta \restriction i) \cdot S_i(\alpha \restriction i), S_i(\beta \restriction i) \cdot sl'_i) \in B_i$. Moreover, since the B_i are total in $\overline{N_j}$ and the $\mathcal{V}_i$ distinguish $\overline{N_j}$, we can remove all occurrences of secrets produced by those events from sl'_i and get $sl''_i = sl'_i \setminus \overline{N_j}$ such that $sl' \in sl''_1 \parallel_S sl''_2$ and $(S_i(\beta \restriction i) \cdot S_i(\alpha \restriction i), S_i(\beta \restriction i) \cdot sl''_i) \in B_i$. We now distinguish the different side conditions.

- *BSBD*: We apply the BD Security properties of LTS_1 and LTS_2 with side condition *BSBD*, respectively, and obtain α'_1 and α'_2 with $(\beta \restriction i) \cdot \alpha'_i \in [\![LTS_i]\!]$, $O_i(\alpha'_i) = O_i(\alpha \restriction i)$, and $S_i(\alpha'_i) = sl''_i$. We then apply the zipping lemma C.1 to obtain α' with $\beta \cdot \alpha' \in [\![LTS]\!]$, $O(\alpha') = O(\alpha)$, and $S(\alpha') = sl'$.

- *EI*$[X]$: Let sl' begin with $getSec(x)$ for some $x \in X \cap (Ev^{sec} \setminus Ev^{obs})$, i.e., $sl' = getSec(x) \cdot sl''$ for some sl''. We perform a case distinction on x:
  - $x = (1, x_1)$ with $x_1 \in X_1 \cap (Ev_1^{sec} \setminus Ev_1^{obs})$. With $sl' \in sl''_1 \parallel_S sl''_2$ we get $sl''_1 = getSec_1(x_1) \cdot sl'''_1$ for some sl'''_1. We apply the BD Security property of LTS_1 with *EI*, obtaining $(\beta \restriction 1) \cdot x'_1 \cdot \alpha'_1 \in [\![LTS_1]\!]$ with $x'_1 \approx_{\mathcal{V}_1} x_1$, $S_1(\alpha'_1) = sl'''_1$, and $O_1(\alpha'_1) = O_1(\alpha_1)$. Moreover, we apply the BD Security property of LTS_2 with *BSBD*, obtaining $(\beta \restriction 2) \cdot \alpha'_2 \in [\![LTS_2]\!]$ with $S_2(\alpha'_2) = sl''_2$ and $O_2(\alpha'_2) = O_2(\alpha_2)$. Finally, we apply Lemma C.1 to merge α'_1 and α'_2 to obtain $\beta \cdot (1, u'_1) \cdot \alpha' \in [\![LTS]\!]$ with $S(\alpha') = sl''$ and $O(\alpha') = O(\alpha)$.
  - $x = (2, x_2)$ with $x_2 \in X_2 \cap (Ev_2^{sec} \setminus Ev_2^{obs})$. Symmetric to the previous case.
  - $x = (x_1, x_2)$ with $x_i \in X_i \cap (Ev_i^{sec} \setminus Ev_i^{obs})$. Since $sl' \in sl''_1 \parallel_S sl''_2$, we obtain $x'_1 \in Ev_1^{sec}$, $x'_2 \in Ev_2^{sec}$, sl'''_1, and sl'''_2 such that $sl''_i = getSec_i(x'_i) \cdot sl'''_i$, $getSec(x) = getSec_1(x'_1) *_S getSec_2(x'_2)$, and either $x'_1 \parallel x'_2$ or $x'_2 \parallel x'_1$. Let $x'_1 \parallel x'_2$ hold (the other case is symmetric). Since $(x_1, x_2) \approx_{\mathcal{V}} (x'_1, x'_2)$ and $(x_1, x_2) \in X$ imply $(x'_1, x'_2) \in X$ due to the assumptions of the theorem, we have $x'_i \in X_i$. This allows us to apply *EI*$[X_i]$ to the component traces to obtain $(\beta \restriction i) \cdot x'''_i \cdot \alpha'_i \in [\![LTS_i]\!]$ with $S_i(\alpha'_i) = sl'''_i$ and $O_i(\alpha'_i) = O_i(\alpha_i)$. If x'''_1 and x'''_2 do not synchronize, we apply *LR*$[\bowtie_{2,1}]$ to adapt x'''_2 to an event that matches x'''_1, as we did in the proof of Lemma C.1. Since $\mathcal{V}_1$ and $\mathcal{V}_2$ have complementary secrets and observations, the resulting composed event (x'''_1, x'''_2) is still $\approx_{\mathcal{V}}$-equivalent to (x_1, x_2) (cf. Definition 3.15), which again implies $(x'''_1, x'''_2) \in X$ due to the assumptions of this theorem. Finally, we apply Lemma C.1 to merge the remaining traces α'_1 and α'_2.

- *LR*$[R]$: Let α begin with $e \in Ev^{sec}$ such that $e \, R \, e'$ and $sl' = getSec(e') \cdot sl''$ for some $e' \in Ev^{sec}$. We perform a case distinction on e:
  - $e = (1, e_1)$. By the construction of R, we obtain $e' = (1, e'_1)$ with $(e_1, e'_1) \in R_1$. We apply the *LR*$[R_1]$ property of LTS_1 in order to obtain

$(\beta \restriction 1) \cdot e_1'' \cdot \alpha_1' \in [\![LTS_1]\!]$ with $e_1'' \approx_{\mathcal{V}_1} e_1'$. Moreover, we apply the *BSBD* property of LTS_2 to obtain $(\beta \restriction 2) \cdot \alpha_2' \in [\![LTS_2]\!]$ where α_2' produces the secrets sl_2''. Again, we apply Lemma C.1 to merge α_1' and α_2'.

- $e = (2, e_2)$. Symmetric to the previous case.
- $e = (e_1, e_2)$. By the construction of R, we obtain $e' = (e_1', e_2')$ with either $e_1' \parallel e_2'$ or $e_2' \parallel e_1'$. We apply the $LR(R_i)$ properties of *both* systems to replace the events, which can then be synchronized to a composed event (e_1', e_2'). Afterwards, we merge the remaining traces using Lemma C.1, as usual.

- $IB[X, Y, Z]$. Let α begin with $z \in Z$, and let sl' begin with $S(\langle x, z\rangle)$ for some $x \in X \cap Ev^{sec}$. We perform a case distinction on z.
  - $z = (1, z_1)$ with $z_1 \in Z_1$. Hence, $\alpha \restriction 1$ begins with z_1. We proceed with a case distinction on x.
    - $x = (1, x_1)$ with $x_1 \in X_1 \cap Ev_1^{sec} \setminus Ev_1^{obs}$. Since sl' begins with $S(\langle x, z\rangle)$, sl_1'' must begin with $S_1(\langle x_1, z_1\rangle)$. This allows us to invoke $IB[X_1, Y_1, Z_1]$ to obtain $(\beta \restriction 1) \cdot x_1' \cdot ys_1' \cdot z_1' \cdot \alpha_1' \in [\![LTS_1]\!]$. Since $x_1' \cdot ys_1' \cdot z_1'$ only consists of local events of LTS_1 (note that $If_1 \subseteq Y_1$ by assumption and $ys_1' \in (N_1 \setminus Y_1)^*$), it can be directly lifted to a trace of the composed system $x' \cdot ys' \cdot z'$ such that $\beta \cdot x' \cdot ys' \cdot z' \in [\![LTS]\!]$, $O(x' \cdot ys' \cdot z') = O(\langle z\rangle)$, and $S(x' \cdot ys' \cdot z') = S(\langle x, z\rangle)$. Finally, we apply the *BSBD* property of LTS_2 to obtain an α_2' producing sl_2'' and invoke Lemma C.1 to merge it with α_1'.
    - $x = (2, x_2)$ with $x_2 \in X_2 \cap Ev_2^{sec} \setminus Ev_2^{obs}$. Symmetric to the previous case.
    - $x = (x_1, x_2)$ with $x_i \in X_i \cap Ev_i^{sec} \setminus Ev_i^{obs}$. Since sl' begins with $getSec(x)$, sl_1'' and sl_2'' must begin with $getSec_1(x_1')$ and $getSec_2(x_2')$, respectively, for some x_i' with $getSec(x) = getSec_1(x_1') *_S getSec_2(x_2')$ and either $x_1' \parallel x_2'$ or $x_2' \parallel x_1'$. In addition to inserting both x_1' (using $IB[X_1, Y_1, Z_1]$) and x_2' (using $EI[X_2]$), we have to *synchronize* them eventually. We proceed with a case distinction on the direction of communication.
      - $x_1' \parallel x_2'$, i.e., x_1' is the output event. We apply $IB[X_1, Y_1, Z_1]$ as above to obtain $(\beta \restriction 1) \cdot x_1'' \cdot ys_1' \cdot z_1' \cdot \alpha_1' \in [\![LTS_1]\!]$. If x_2' synchronizes with x_1'', we apply $EI[X_2]$ to obtain $(\beta \restriction 2) \cdot x_2'' \cdot \alpha_2' \in [\![LTS_2]\!]$. Since $x_2' \approx_{\mathcal{V}_2} x_2''$, the latter is guaranteed to synchronize with x_1'' as well, due to the second condition of Definition 3.7. If, however, the original x_2' does *not* synchronize with x_1'', we note that there must be *some* other x_2''' with $x_1'' \parallel x_2'''$ (due to the fourth condition of Definition 3.15 together with $x_1' \parallel x_2'$ and $x_1' \approx_{\mathcal{V}_1} x_1''$). Hence, $x_2' \bowtie_{2,1} x_2'''$, and since B_2 includes $\bowtie_{2,1}$, we can replace $getSec_2(x_2')$ by $getSec_2(x_2''')$ at the beginning of sl_2''. We then

apply $EI[X_2]$ as above and are guaranteed to obtain a x_2'' that synchronizes with x_1''.

· $x_2' \parallel x_1'$. The proof proceeds like in the previous case. The only difference is that we now first invoke $EI[X_2]$ to obtain the output event x_2''. Depending on whether x_1' synchronizes with x_2'' or not, we now possibly adapt sl_1'' analogous to above before applying $IB[X_1, Y_1, Z_1]$ to obtain a x_1'' that is guaranteed to synchronize with x_2''.

Finally, we close the zipper up to z_1 analogous to the case $x = (1, x_1)$, and conclude the proof by applying Lemma C.1 to the remaining traces α_1' and α_2'.

– $z = (2, z_2)$ with $z_2 \in Z_2$. Symmetric to the previous case.

– $z = (z_1, z_2)$ with $z_i \in Z_i$. Hence, each $\alpha \upharpoonright i$ begins with z_i. We proceed with a case distinction on x.

∗ $x = (1, x_1)$ with $x_1 \in X_1 \cap Ev_1^{sec} \setminus Ev_1^{obs}$. We proceed like in the corresponding case above (for $z = (1, z_1)$), up to the point where we are about to apply *BSBD*. Before that, we use either $LR[\bowtie_{1,2}]$ or $LR[\bowtie_{2,1}]$ to synchronize z_1' and z_2, if necessary, depending on which one is the output event. Since the resulting composed event (z_1'', z_2'') is $\approx_{\mathcal{V}}$-equivalent to (z_1, z_2) due to the fact that the views have complementary secrets and observations (cf. Definition 3.15), the assumptions of the theorem imply that $(z_1'', z_2'') \in Z$. Next we invoke *BSBD* of LTS_2 *after* the (possibly adapted) event z_2'', obtaining $(\beta \upharpoonright 2) \cdot z_2'' \cdot \alpha_2' \in [\![LTS_2]\!]$, where $S_2(z_2' \cdot \alpha_2') = sl_2''$. Finally, we apply Lemma C.1 to merge α_1' and α_2' to a composed trace producing the remaining secrets in sl' (after $S(\langle u, n\rangle)$).

∗ $x = (2, x_2)$ with $x_2 \in X_2 \cap Ev_2^{sec} \setminus Ev_2^{obs}$. Symmetric to the previous case.

∗ $x = (x_1, x_2)$ with $x_i \in X_i \cap Ev_i^{sec} \setminus Ev_i^{obs}$. We proceed analogous to the corresponding case for $z = (1, z_1)$ above, but now using $IB[X_2, Y_2, Z_2]$ instead of $EI[X_2]$. Moreover, we now have to synchronize z_1' and z_2', possibly using $LR(\bowtie_{i,j})$ as above. Afterwards, we merge the prefixes of the component traces $(\beta \upharpoonright i) \cdot x_i'' \cdot ys_i \cdot z_i'' \in [\![LTS_i]\!]$ to a composed trace $\beta \cdot x'' \cdot ys_1 \cdot ys_2 \cdot z'' \in [\![LTS]\!]$, and merge the remaining traces α_1'' and α_2'' using Lemma C.1.

• $DB[X, Y, Z]$. Let α begin with $x \cdot ys \cdot z$ with $x \in X \cap Ev^{sec}$, $ys \in (N \setminus Y)^*$, and $z \in Z$, and let sl' begin with $S(\langle z\rangle)$. We perform a case distinction on x.

– $x = (1, x_1)$ with $x_1 \in X_1 \cap Ev_1^{sec} \setminus Ev_1^{obs}$. We perform a case distinction on z.

∗ $z = (1, z_1)$ with $z_1 \in Z_1$. Hence, $\alpha \upharpoonright 1$ begins with $x_1 \cdot (ys \upharpoonright 1) \cdot z_1$. We invoke $DB[X_1, Y_1, Z_1]$ to obtain $(\beta \upharpoonright 1) \cdot ys_1' \cdot z_1' \cdot \alpha_1' \in [\![LTS_1]\!]$,

where α_1' produces the remaining observations of $\alpha \restriction 1$ and sl_1'' after z_1. We invoke *BSBD* to obtain $(\beta \restriction 2) \cdot \alpha_2'$, where α_2' produces sl_2''. Finally, we apply Lemma C.1 to merge α_1' and α_2'.

  - $z = (2, z_2)$ with $z_2 \in Z_2$. Hence, $\alpha \restriction 2$ begins with $(ys \restriction 2) \cdot z_2$. This subsequence of α already satisfies the condition on the desired suffix α', so we simply "pull it backwards" past the subsequence of α produced by LTS_1. This is possible since all events in $x \cdot ys \cdot z$ are local and do not involve communication (note that $If_i \subseteq Y_i$ by assumption, hence ys does *not* contain interface events), so we can change their interleaving arbitrarily. We get $\beta \cdot (ys \restriction 2) \cdot z \in [\![LTS]\!]$. We invoke *BSBD* on the remaining component traces, obtaining α_i' that produce the remaining secrets in sl_i'', and then merge those α_i' using Lemma C.1.

  - $z = (z_1, z_2)$ with $z_i \in Z_i$. As in the case $z = (1, z_1)$, we apply $DB[X_1, Y_1, Z_1]$ to obtain $(\beta \restriction 1) \cdot ys_1' \cdot z_1' \cdot \alpha_1' \in [\![LTS_1]\!]$. As in the case $z = (2, z_2)$, we pull $ys \restriction 2$ backwards and invoke *BSBD* after z_2, obtaining α_2'. We get $\beta \cdot ys_1' \cdot (ys \restriction 2) \in [\![LTS]\!]$. If z_1' does not synchronize with z_2, we use $LR[\bowtie_{i,j}]$ to adapt the input event, before we apply Lemma C.1 to conclude the proof as usual.

- $x = (2, x_2)$ with $x_2 \in X_2 \cap Ev_2^{sec} \setminus Ev_2^{obs}$. Symmetric to the previous case.

- $x = (x_1, x_2)$ with $x_i \in X_i \cap Ev_i^{sec} \setminus Ev_i^{obs}$. We perform a case distinction on z.

  - $z = (1, z_1)$ with $z_1 \in Z_1$. As in the corresponding case above (for $x = (1, x_1)$), we invoke $DB[X_1, Y_1, Z_2]$ to obtain $(\beta \restriction 1) \cdot ys_1' \cdot z_1' \cdot \alpha_1'$ as a possible trace of LTS_1, and we invoke *BSBD* to obtain an α_2' producing sl_2'' that we merge with α_1' using Lemma C.1.

  - $z = (2, z_2)$ with $z_2 \in Z_2$. Symmetric to the previous case: We invoke $DB[X_2, Y_2, Z_2]$ on the trace of LTS_2, and *BSBD* on the trace of LTS_1.

  - $z = (z_1, z_2)$ with $z_i \in Z_i$. We now apply $DB[X_i, Y_i, Z_i]$ on both traces, obtaining $(\beta \restriction i) \cdot ys_i' \cdot z_i' \cdot \alpha_i' \in [\![LTS_i]\!]$. Since the ys_i' contain only local events, we simply include them one after the other in the merged trace. If z_1' and z_2' do not synchronize, we apply $LR[\bowtie_{i,j}]$ as usual, followed by Lemma C.1 to merge α_1' and α_2'. □

C.3. n-ary Composition and Case Study

Proofs of the theorems about n-ary composition in Section 4.3, as well as all specifications, theorems and proofs presented as part of the case study in Chapter 4 have been formalized in the interactive theorem prover Isabelle/HOL in collaboration with Andrei Popescu. The theory files are available for download from the website

of CoSMeDis at `http://andreipopescu.uk/CoSMeDis.html`. The files are accompanied by a README file that explains which theorems are formalized in which files.

We refer to that formalization for details and only summarize the proofs for the generic theorems in Section 4.3 here.

Proof sketch of Theorem 4.1. We consider a sub-network Net' containing the nodes $N' \cup \{k\} \subseteq N$ w.r.t. the synchronization relations $(\|_{i,j})_{i,j \in N' \cup \{k\}}$ restricted to nodes in the sub-network. We proceed by finite set induction on N'. In the base case, the security of the network consisting only of the node k follows immediately from the local security property of that node. In the inductive step, we add a new node i. In order to compose the network $N' \cup \{k\}$ with the node i, we apply a binary composition result [BPPR17, Theorem 1] that is similar to Theorem 3.4, but makes use of the simplifying assumptions of Section 4.3. The direct result of applying the binary theorem is a security property for the canonical composition of sub-network and new node, with events of the form, for example, $((k, e), e')$ for a communication event between secret issuer and new node with $e \|_{k,i} e'$. We use the translation theorem 4.2 with suitable bijections to translate the composed system and security property back to the canonical form for the network with nodes $N' \cup \{i, k\}$. The overall security property follows from setting $Net' = Net$. □

Proof sketch of Theorem 4.2. Let $t_2 \in [\![LTS_2]\!]$ and $(S_{\mathcal{V}_2}(t_2), sl'_2) \in B'_2$. Using the first and third conditions of Definition 4.8 and an inductive argument, we obtain $t_1 \in [\![LTS_1]\!]$ with $\mathrm{map}_{\pi_{Ev}}(t_1) = t_2$. With Definition 4.9, we get the fact that $\mathrm{map}_{\pi_S}(S_{\mathcal{V}_1}(t_1)) = S_{\mathcal{V}_2}(t_2)$. Using the second assumption of the theorem, we get $(S_{\mathcal{V}_1}(t_1), sl'_1) \in B_1$ with $\mathrm{map}_{\pi_S}(sl'_1) = sl'_2$. We use the BD Security property of LTS_1 to obtain t'_1 with the same observations as t_1 and the secrets sl'_1. With the first and second conditions of Definition 4.8, we get $t'_2 = \mathrm{map}_{\pi_{Ev}}(t'_1) \in [\![LTS_2]\!]$, and with Definition 4.9 we get $O_{\mathcal{V}_2}(t'_2) = O_{\mathcal{V}_2}(t_2)$ and $S_{\mathcal{V}_2}(t'_2) = sl'_2$.

For proving the translation of side conditions, we additionally assume that π_{Ev} and π_S are bijections. Hence, $\mathrm{map}_{\pi_{Ev}}$ and map_{π_S} are also bijections. Moreover, they respect trace concatenation in the sense that $\mathrm{map}_{\pi_{Ev}}(\beta \cdot \alpha) = \mathrm{map}_{\pi_{Ev}}(\beta) \cdot \mathrm{map}_{\pi_{Ev}}(\alpha)$, and analogously for map_{π_S}. We follow the same proof structure as for plain BD Security above, but preserving backwards-strictness: Fix β_2, α_2, and sl'_2 with

- $\beta_2 \cdot \alpha_2 \in [\![LTS_2]\!]$,
- $(S_{\mathcal{V}_2}(\beta_2) \cdot S_{\mathcal{V}_2}(\alpha_2), S_{\mathcal{V}_2}(\beta_2) \cdot sl'_2) \in B_2$, and
- $P_2(\beta_2, \alpha_2, sl'_2)$.

Let $\beta_1 = \mathrm{map}_{\pi_{Ev}^{-1}}(\beta_2)$, $\alpha_1 = \mathrm{map}_{\pi_{Ev}^{-1}}(\alpha_2)$, and $sl'_1 = \mathrm{map}_{\pi_S^{-1}}(sl'_2)$. We use the assumptions and Definitions 4.8 and 4.9 as above together with the definition of P_2 to obtain

- $\beta_1 \cdot \alpha_1 \in [\![LTS_1]\!]$,

- $(S_{\mathcal{V}_1}(\beta_1) \cdot S_{\mathcal{V}_1}(\alpha_1), S_{\mathcal{V}_1}(\beta_1) \cdot sl_1') \in B_1$, and
- $P_1(\beta_1, \alpha_1, sl_1')$.

This allows us to use the BD Security property of LTS_1 with side condition (P_1, Q_1) to obtain α_1' with $\beta_1 \cdot \alpha_1' \in [\![LTS_1]\!]$, $O_{\mathcal{V}_1}(\alpha_1') = O_{\mathcal{V}_1}(\alpha_1)$, $S_{\mathcal{V}_1}(\alpha_1') = S_{\mathcal{V}_1}(\alpha_1')$, and $Q_1(\beta_1, \alpha_1, \alpha_1')$. Again using the assumptions as above, this time together with the definition of Q_2, it follows that $\alpha_2' = \mathrm{map}_{\pi_{Ev}}(\alpha_1')$ is a suitable witness with $\beta_2 \cdot \alpha_2' \in [\![LTS_2]\!]$, $O_{\mathcal{V}_2}(\alpha_2') = O_{\mathcal{V}_2}(\alpha_2)$, $S_{\mathcal{V}_2}(\alpha_2') = S_{\mathcal{V}_2}(\alpha_2')$, and $Q_2(\beta_2, \alpha_2, \alpha_2')$. □

Proof sketch of Theorem 4.3. By applying Theorem 4.1 to obtain the canonical security property for the network, and then applying Theorem 4.2 choosing the secret translation function π_S that extracts the secret of the issuer, and the identity for the other translation functions. □

C.4. Workflow Systems

Proof of Theorem 5.1. We prove the theorem by finite set induction on $\mathcal{A}$. For this purpose, we consider a subsystem

$$LTS_{\mathcal{A}'} = (\|_{a \in \mathcal{A}'} LTS_a) \,\|\, LTS_{P_W}$$

for an arbitrary but fixed $\mathcal{A}' \subseteq \mathcal{A}$. Let

$$N^P_{\mathcal{A}'} = \left\{ Send_a(b,m) \mid a \in \mathcal{A} \setminus \mathcal{A}' \wedge Send_a(b,m) \notin Ev^{sec}_a \cup Ev^{obs}_a \right\}$$

denote the set of *Send* events that have to be secret for the platform, because they are neutral for activities that still have to be merged. We consider the view $\mathcal{V}_{\mathcal{A}'} = (Ev^{obs}_{\mathcal{A}'}, \mathrm{id}, Ev^{sec}_{\mathcal{A}'}, \mathrm{id})$ and bound $B_{\mathcal{A}'}$ with

$$\begin{aligned}
Ev^{obs}_{\mathcal{A}'} &= \bigcup_{a \in \mathcal{A}'} Ev^{obs}_a \cup \bigcup_{a \in \mathcal{A} \setminus \mathcal{A}'} Ev^{obs}_a \cap Ev_P \\
Ev^{sec}_{\mathcal{A}'} &= \bigcup_{a \in \mathcal{A}'} Ev^{sec}_a \setminus N_a \cup \bigcup_{a \in \mathcal{A} \setminus \mathcal{A}'} ((Ev^{sec}_a \cap Ev_P) \setminus N_a) \cup N^P_{\mathcal{A}'} \\
B_{\mathcal{A}'} &= \big\{ (sl, sl') \mid \forall a \in \mathcal{A}'. (sl \restriction Ev^{sec}_a, sl' \restriction Ev^{sec}_a) \in B_a \wedge \\
&\qquad\qquad sl \equiv_{Ev^{obs}_{\mathcal{A}'}} sl' \wedge causal(sl') \big\}
\end{aligned}$$

In order to prove acceptance of neutral events without placing too many requirements on the components LTS_a, we define another bound that is total in $N^P_{\mathcal{A}'}$, but does not require altering any other secret events (note that Definition 3.11 only requires *some* bound that is total in neutral events).

$$B^N_{\mathcal{A}'} = \big\{ (t, t') \mid t, t' \in (Ev^{sec}_{\mathcal{A}'})^*.\ (t \setminus N^P_{\mathcal{A}'}) = (t' \setminus N^P_{\mathcal{A}'}) \big\}$$

We show that $LTS_{\mathcal{A}'}$

1. satisfies BD Security w.r.t. $\mathcal{V}_{\mathcal{A}'}$ and $B_{\mathcal{A}'}$,

2. satisfies backwards-strict BD Security, eager insertion of $N^P_{\mathcal{A}'}$, and insertion and deletion of $N^P_{\mathcal{A}'}$ before $Ev^{obs}_{\mathcal{A}'} \setminus Ev^{sec}_{\mathcal{A}'}$ without Ev_P in between w.r.t. $\mathcal{V}_{\mathcal{A}'}$ and $B^N_{\mathcal{A}'}$, and

3. if case c) of the third assumption of the theorem holds, then it supports eager insertion of $Ev^{sec}_{\mathcal{A}'} \setminus Ev^{obs}_{\mathcal{A}'}$ as well as insertion and deletion of $Ev^{sec}_{\mathcal{A}'} \setminus Ev^{obs}_{\mathcal{A}'}$ before $Ev^{obs}_{\mathcal{A}'} \setminus Ev^{sec}_{\mathcal{A}'}$ without Ev_P in between w.r.t. $\mathcal{V}_{\mathcal{A}'}$ and $B^N_{\mathcal{A}'}$.

The proof proceeds by finite set induction on $\mathcal{A}'$.

In the base case $\mathcal{A}' = \emptyset$, we have to prove the security properties of the platform itself. It is easy to see that it supports the arbitrary insertion, deletion, and replacement of secret sending and receiving events, as long as causality and observable messages are preserved, which is encoded in the bound $B_\emptyset$. In particular, for the second proof goal, the backwards-strict and eager insertion of the *Send* events in $N^P_{\mathcal{A}'}$ is always possible at any position, including before $Ev^{obs}_{\mathcal{A}'} \setminus Ev^{sec}_{\mathcal{A}'}$. This does not require any changes to the rest of the trace, since the platform does not guarantee delivery of sent messages in order, or at all (we consider a variant with an ordered delivery guarantee in Chapter 6). Moreover, for the third proof goal above, we can always swap a secret and a succeeding observable event (or vice versa) without changing the trace before or after those events. Neutral events in between can be moved after the pair of events (delaying the receipt of messages is allowed, again due to the lack of delivery guarantees). Causality is not affected by this reordering of secret and observable events, since it does not swap matching $Send_a(b, m)$ and $Recv_b(a, m)$ pairs; the former is observable iff the latter is, and here only swapping of events in $(Ev^{sec}_P \setminus Ev^{obs}_P)$ and $(Ev^{obs}_P \setminus Ev^{sec}_P)$, respectively, is required. Hence, the third proof goal only requires the swapping of communication events between different activities or with different content, which is supported by the platform.

In the inductive step, we compose $LTS_{\mathcal{A}'}$ with an activity component LTS_a for $a \in \mathcal{A} \setminus \mathcal{A}'$. The views of the platform and the activity are aligned on events in Ev_a, with the exception of pairs of *Send* and *Recv* events that become neutral after composition, i.e., where the *Send* event is neutral for the sending activity; in that case, if a is the sending activity, then the *Send* event is in $N^P_{\mathcal{A}'}$ and is neutral for LTS_a but secret for $LTS_{\mathcal{A}'}$, whereas if a is the receiving activity, then the *Recv* event is in N_a and secret for LTS_a but neutral for $LTS_{\mathcal{A}'}$. Moreover, the bounds of both the subsystem and the new component are total in those respective events. Hence, the bounds of the subsystem and the additional component are composable, and the first condition of view composability (cf. Definition 3.7) is satisfied. The second condition, as well as the conditions of complementary secrets and observations (Definition 3.15) and well-definedness (Definition 3.20) are trivially satisfied, because the relations $\approx_{\mathcal{V}_i}$ are the identity and the relations $\bowtie_{i,j}$ are empty. The subsystem $LTS_{\mathcal{A}'}$ and the component LTS_a accept each other's neutral events, because, first, they satisfy *EI* for their respective neutral events, and second, $LTS_{\mathcal{A}'}$ silently accepts $N^P_{\mathcal{A}'}$ before $Ev^{obs}_{\mathcal{A}'} \cup Ev^{sec}_{\mathcal{A}'}$ via the induction hypothesis, which covers the interface between $LTS_{\mathcal{A}'}$

and LTS_a. Hence, the first condition (mutual acceptance of neutral events) of Lemma 3.3 is satisfied. The second condition (mutual acceptance of secrets and observations) is trivially satisfied, because the relations $\bowtie_{i,j}$ are empty. The third condition is satisfied because flexible scheduling of secrets and observations follows from a case distinction on the third assumption of the theorem:

a) If $Ev_a^{obs} \subseteq Ev_a^{sec}$ for all $a \in \mathcal{A}$, then $Ev_a^{obs} \setminus Ev_a^{sec} = \emptyset$ as well as $Ev_{\mathcal{A}'}^{obs} \setminus Ev_{\mathcal{A}'}^{sec} = \emptyset$ and flexible scheduling of secrets and observations trivially holds for both LTS_a and $LTS_{\mathcal{A}'}$.

b) If $Ev_a^{sec} \setminus N_a \subseteq Ev_a^{obs}$ for all $a \in \mathcal{A}$, then $Ev_a^{sec} \setminus Ev_a^{obs} = N_a$, and if $N_a \neq \emptyset$, then LTS_a supports insertion and deletion of N_a before $Ev_a^{obs} \setminus Ev_a^{sec}$ without communication events in between w.r.t. $\mathcal{V}_a$ and Id due to the second assumption of the theorem (note that either B_a is total in N_a and therefore reflexive, or $N_a = \emptyset = Ev_a^{sec} \setminus Ev_a^{obs}$, so flexible scheduling of secrets and observations in LTS_a holds trivially). Flexible scheduling of secrets and observations for $LTS_{\mathcal{A}'}$ follows via an analogous argument: $Ev_a^{sec} \setminus N_a \subseteq Ev_a^{obs}$ for all $a \in \mathcal{A}$ implies $Ev_{\mathcal{A}'}^{sec} \setminus N_{\mathcal{A}'}^P \subseteq Ev_{\mathcal{A}'}^{obs}$. Hence, $Ev_{\mathcal{A}'}^{sec} \setminus Ev_{\mathcal{A}'}^{obs} = N_{\mathcal{A}'}^P$, and $LTS_{\mathcal{A}'}$ supports insertion and deletion of $N_{\mathcal{A}'}^P$ before $Ev_{\mathcal{A}'}^{obs} \setminus Ev_{\mathcal{A}'}^{sec}$ w.r.t. $\mathcal{V}_{\mathcal{A}'}$ and $B_{\mathcal{A}'}^N$ via the induction hypothesis.

c) Otherwise, flexible scheduling of secrets and observations is supported by LTS_a via assumption 3c) of the theorem, and by $LTS_{\mathcal{A}'}$ via the induction hypothesis.

This allows us to apply Theorem 3.4 to obtain BD Security of $LTS_{\mathcal{A}'} \parallel LTS_a$ w.r.t. the composition of $\mathcal{V}_{\mathcal{A}'}$ with $\mathcal{V}_a$ and $B_{\mathcal{A}'}$ with B_a, respectively. We use Theorem 4.2 to translate this to the desired BD Security property of $LTS_{\mathcal{A}' \cup \{a\}}$ w.r.t. $\mathcal{V}_{\mathcal{A}' \cup \{a\}}$ and $B_{\mathcal{A}' \cup \{a\}}$ using a bijective mapping, as we have done in the proof of Theorem 3.5. For the second and third proof goals, we have to show the side conditions w.r.t. the *remaining* neutral events $N_{\mathcal{A}' \cup \{a\}}^P = N_{\mathcal{A}'}^P \setminus Ev_a$.

We use Theorem 3.7 to show the second proof goal using corresponding properties of $LTS_{\mathcal{A}'}$ and LTS_a, respectively. For the former, we use the second part of the induction hypothesis. For the latter, we use the second assumption of the theorem to show that LTS_a satisfies backwards-strict BD Security, eager insertion of N_a and insertion and deletion of N_a before $Ev_a^{obs} \setminus Ev_a^{sec}$ w.r.t. $\mathcal{V}_a$ and

$$B_a^N = \left\{(t, t') \mid t, t' \in (Ev_a^{sec})^*.\ (t \setminus N_a) = (t' \setminus N_a)\right\}$$

The preconditions of Theorem 3.7 hold:

- We already showed the preconditions for well-behavedness (Lemma 3.3) above.
- B_a^N and $B_{\mathcal{A}'}^N$ are total in N_a and $N_{\mathcal{A}'}^P$, respectively, and the $\bowtie$ relations are empty since the observation and secret extracting functions are the identity.
- The constraints in Table 3.2 are satisfied, since the event sets X_i here are exactly the respective neutral, and therefore non-observable, events, and the sets Y_i include the interface events.

- $\mathcal{V}_{\mathcal{A}'\cup\{a\}}$ distinguishes $N^P_{\mathcal{A}'\cup\{a\}}$ as well as $Ev^{obs}_{\mathcal{A}'\cup\{a\}} \setminus Ev^{sec}_{\mathcal{A}'\cup\{a\}}$ because $\approx_{\mathcal{V}_{\mathcal{A}'\cup\{a\}}}$ is the identity.

- $LTS_{\mathcal{A}'}$ satisfies the required combinations of properties via the induction hypothesis. For LTS_a, if $N_a \neq \emptyset$, then these properties hold due to the second assumption of the theorem (note that B_a is total in N_a, so $B^N_a \subseteq B_a$), and if $N_a = \emptyset$, then the insertion and deletion properties are trivially satisfied, and B^N_a collapses to the identity, so backwards-strict BD Security trivially holds as well.

We obtain the corresponding security properties of $LTS_{\mathcal{A}'} \parallel LTS_a$ w.r.t. the events in $N^P_{\mathcal{A}'\cup\{a\}}$ and the backwards-strict bounds composition of $B^N_{\mathcal{A}'}$ and B^N_a via Theorem 3.7. We use Theorem 4.2 using a bijective mapping as above to obtain the required security properties w.r.t. $N^P_{\mathcal{A}'\cup\{a\}}$ and $B^N_{\mathcal{A}'\cup\{a\}}$.

For the third proof goal, if case c) of the third assumption of the theorem holds, then we use the security properties as given there for LTS_a, but restricted to the history-independent bound $B^N_a \subseteq B_a$ for backwards-strict composition. For $LTS_{\mathcal{A}'}$, we use the induction hypothesis, and then use Theorem 3.7 as for the second goal above.

The conclusion of the theorem follows by setting $\mathcal{A}' = \mathcal{A}$. □

Proof sketch of Theorem 5.2. Let

$$LTS'_W = \left(\|_{a'\in\mathcal{A}_W\setminus\{a\}} LTS_{a'}\right) \parallel LTS_{P_W} \parallel LTS_{Proxy(a,W')}$$

denote the subsystem of the workflow W with a replaced by the proxy. Let $\mathcal{V}_a$ be the local view on a for d w.r.t. $\mathcal{P}[a \leftarrow \mathcal{P}']$. The proxy merely forwards input and output data from the activities in W to activities in W' and vice versa. Since the classifications of shared data items coincide between the two workflows, and the classification of entry and exit activities of W' are equal to that of a, the proxy does not declassify any information except the static knowledge that it sends data items and triggers according to the workflow specification of W', and when it sends a data item then its value corresponds to the value last received. Capturing this in a declassification bound, $B_{Proxy(a,W')}$, we can show that $LTS_{Proxy(a,W')}$ satisfies BD Security w.r.t. $\mathcal{V}_a$ and $B_{Proxy(a,W')}$.

Let $\mathcal{V}'_W$ denote the global view on W for d w.r.t. $\mathcal{P}[a \leftarrow \mathcal{P}']$, and $B'_{\mathcal{A}}$ denote the bound for W with the component bound of a replaced by $B_{Proxy(a,W')}$. We obtain the security of LTS'_W w.r.t. $\mathcal{V}'_W$ and $B'_{\mathcal{A}}$ via Theorem 5.1. Its assumptions are satisfied since each $LTS_{a'}$ satisfies BD Security w.r.t. $\mathcal{V}_{a'}$ and $B_{a'}$, and for the views we assume in this chapter we have $N_{a'} = \emptyset$ as well as $Ev^{obs}_{a'} \subseteq Ev^{sec}_{a'} \cap Ev_{P_W}$.

We use Theorem 3.4 to obtain a security property for $LTS'_W \parallel LTS_{W'}$. Since there are no neutral events at the interface, the secret producing functions are the identity, and $Ev^{obs} \subseteq Ev^{sec}$ holds for both subsystem views, the preconditions of the theorem are trivially satisfied. Finally, we translate $LTS'_W \parallel LTS_{W'}$ to $LTS_{W[a\leftarrow W']}$ using a bijective mapping, as we have done it in the proofs of Theorems 3.5 and 5.1. □

C.5. Safety Properties

Proof of Lemma 6.1. Let $t \in [\![LTS \lhd LTS_M]\!]$ be an arbitrary but fixed trace of the monitored system. For the composition operator $\lhd$ we use in this chapter, this implies $(t \restriction Ev) \in [\![LTS]\!]$ and $(t \restriction Ev_M) \in [\![LTS_M]\!]$. Moreover, $t \in (Ev \cup Ev_M)^*$ holds, and since P is a property on Ev and Ev_M is a relevant set of events for P, we have $Ev_M \subseteq Ev$, so $(t \restriction Ev) = t$ and therefore $t \in [\![LTS]\!]$. Furthermore, LTS_M enforces P, so $[\![LTS_M]\!] \subseteq P$ holds, which implies $(t \restriction Ev_M) \in P$. Since Ev_M is a relevant set of events for P, the latter implies $t \in P$. Together with $t \in [\![LTS]\!]$ we get $t \in [\![LTS]\!] \cap P$. □

Proof of Lemma 6.2. Let $\mathcal{V} \in \mathcal{V}\mathbf{s}$. Since there are no neutral events in $\mathcal{V}_M(\mathcal{V})$, the conditions about acceptance of neutral events are trivially satisfied.

The monitor also trivially satisfies BD Security: If $Ev_{SoD}^{a,a'} \subseteq Ev_{\mathcal{V}}^{obs} \setminus Ev_{\mathcal{V}}^{sec}$, then there are no secrets, and if $Ev_{SoD}^{a,a'} \subseteq Ev_{\mathcal{V}}^{sec}$, then $S_{\mathcal{V}_M(\mathcal{V})}$ is the identity on monitor traces, and due to the definition of $B_{SoD}^{a,a'}(\mathcal{V})$, for every $(t, t') \in B_{SoD}^{a,a'}(\mathcal{V})$, we can just use t' as the witness for BD Security.

Since $\mathcal{V}$ distinguishes Ev_a, $Ev_{a'}$, and Ev_u for $u \in U$, $e \approx_{\mathcal{V}} e'$ implies that e and e' belong to the same activity and user, so the monitor accepts one iff it accepts the other in any given state. Hence, the monitor supports local replacement via $\approx_{\mathcal{V}}$. □

Proof of Theorem 6.3. As mentioned in the text, we ignore the original classification of input and output events of the target system, and consider all events in Ev_M to be outputs of the target system and inputs of the monitor, with the exception of the events in N_M, which we classify the other way around. We use secret and observation composition operators $*_S$ and $*_O$ that simply copy the secret or observation of the target system.

We prove well-behavedness of $LTS \lhd LTS_M$ using Lemma 3.3. Since G is composable (cf. Definition 6.8), $\mathcal{V}$ distinguishes Ev_M, N_M, and $Ev_{\mathcal{V}}^{sec} \cap Ev_{\mathcal{V}}^{obs}$, so $\mathcal{V}$, $\mathcal{V}_M(\mathcal{V})$, and $\mathcal{V}'$ are well-defined for the respective systems with the above choice of input and output events. Moreover, $\mathcal{V}$ and $\mathcal{V}_M(\mathcal{V})$ are composable: The first condition of Definition 3.7 is satisfied because the views agree on the classification of events at the interface as observable and/or secret, with the exception of events that are neutral for one of the components, which are secret for the other component (due to Definitions 6.8 and 6.5, respectively). The second condition is satisfied because the view-equivalences are the identity on input events:

- For observable and non-secret input events, $\mathcal{V}$ is invertible due to the assumption that G is composable (Definition 6.8).
- Events that are secret for both the monitor and the target system are considered inputs of the monitor, and the secret extraction function in $\mathcal{V}_M(\mathcal{V})$ is the identity.
- Events that are neutral for the target system are secret inputs for the monitor, and the secret extraction function is again the identity.

- Events that are neutral for the monitor are secret inputs for the target system, and $\mathcal{V}$ is again invertible on those events due the composability of G.

Moreover, LTS accepts the neutral events of $\mathcal{V}_M(\mathcal{V})$ due to the assumptions of the theorem, and LTS_M accepts the neutral events of $\mathcal{V}$ because it provides G (cf. Definition 6.6). If both target system and monitor have neutral events, then also by Definition 6.6 the monitor silently accepts the target system's neutral events before $Ev_M \setminus (N_\mathcal{V} \cup N_M)$, covering the interface between monitor and target system (note that pairs of communication events where one is neutral don't have to be covered to satisfy Definition 3.10). Hence, LTS and LTS_M accept each other's neutral events. Mutual acceptance of secrets and observations trivially holds, due to the already observed fact that the view-equivalences are the identity on input events. Finally, LTS and LTS_M are either secret- or observation-polarized at the target system since G is composable (Definition 6.8). Hence, the composition of LTS and LTS_M is well-behaved. After applying Theorem 3.4 w.r.t. $*_S$ and $*_O$ that copy the secrets and observations of the target system (and then translating composed events of the form $(1, e)$, $(2, e)$, or (e, e) back to e using a bijective mapping as in previous chapters), we obtain BD Security of $LTS \lhd LTS_M$ w.r.t. $\mathcal{V}'$ and B', which are merely reformulations of the canonical compositions of $\mathcal{V}$ with $\mathcal{V}_M(\mathcal{V})$ and B with B_M. □

Proof of Lemma 6.4. Fix $\mathcal{V} \in \mathcal{V}\mathbf{s}^{\mathrm{obs}}$. LTS_M trivially satisfies BD Security w.r.t. $\mathcal{V}_M(\mathcal{V})$ and Id, since this does not require any change of secret information. The other conditions of Definition 6.6 are trivially satisfied as well, because there are no neutral events at the interface and $\mathcal{V}$ is invertible on Ev_M. Hence, LTS_M provides G^{obs}. Moreover, G^{obs} is composable: $(Ev^{sec}_\mathcal{V} \cap Ev_M) \setminus N_M \subseteq Ev^{obs}_\mathcal{V}$ holds because $Ev^{sec}_\mathcal{V} \cap Ev_M = \emptyset$, the conditions involving neutral events are trivially satisfied because there are none, and $\mathcal{V}$ is invertible on Ev_M and distinguishes Ev_M and $Ev^{sec}_\mathcal{V} \cap Ev^{obs}_\mathcal{V}$ by assumption. Hence, we can apply Theorem 6.3 to obtain BD Security of $LTS \lhd LTS_M$ w.r.t. $\mathcal{V}'$ and B'. Since $N_M = \emptyset$, we have $\mathcal{V}' = \mathcal{V}$. Moreover, since $Ev^{sec}_{\mathcal{V}_M(\mathcal{V})} = \emptyset$, we have $B' = B$. □

Lemma C.2. *Let the preconditions of Theorem 6.3 hold. Then $B' \subseteq B \parallel^{BS}_B B_M$ holds.*

Proof sketch. Let $(sl, sl') \in B'$. In order to show that $(sl, sl') \in B \parallel^{BS}_B B_M$, we have to find a suitable backwards-strict decomposition of sl' w.r.t. given $\beta \in \beta_1 \parallel_S \beta_2$, $\alpha \in \alpha_1 \parallel_S \alpha_2$, $sl = \beta \cdot \alpha$, and $sl' = \beta \cdot \alpha'$. We note that with the particular composition setup we have here, the decomposition of sl and sl' is fully determined up to neutral events and $\mathcal{V}$-equivalence; we can show the following lemma for any trace t by induction:

$$S_{\mathcal{V}'}(t) \in sl_0 \parallel_S sl_M \longleftrightarrow (sl_0 \setminus getSec_\mathcal{V}(N_M)) = S_{\mathcal{V}'}(t) \wedge (sl_M \setminus N_\mathcal{V}) \approx_\mathcal{V} (t \restriction (Ev^{sec}_{\mathcal{V}_M(\mathcal{V})} \setminus N_\mathcal{V}))$$

In particular, secret sequences of the monitored system simply result from secret sequences of the original system by removing secrets originating from events that are neutral for the monitor (note that $\mathcal{V}$ distinguishes N_M, since $\mathcal{V} \in \mathcal{V}\mathbf{s}$ and the monitor guarantee is composable). Let t_β and t_α be traces such that $S_{\mathcal{V}'}(t_\beta) = \beta$ and $S_{\mathcal{V}'}(t_\alpha) = \alpha$. From the definition of B', we obtain t' with $(S_{\mathcal{V}'}(t_\beta \cdot t_\alpha), S_{\mathcal{V}'}(t')) \in B$, and $((t_\beta \cdot t_\alpha) \upharpoonright Ev^{sec}_{\mathcal{V}_M(\mathcal{V})}, t' \upharpoonright Ev^{sec}_{\mathcal{V}_M(\mathcal{V})}) \in B_M$, and $S_{\mathcal{V}'}(t') = sl'$. Using the latter and $sl' = \beta \cdot \alpha'$, we can decompose t' into $t'_\beta \cdot t'_\alpha$ with $S_{\mathcal{V}'}(t'_\beta) = \beta$, $S_{\mathcal{V}'}(t'_\alpha) = \alpha'$. From the above lemma and $\beta \in \beta_1 \parallel_S \beta_2$, we get the fact that $S_{\mathcal{V}'}(t_\beta) = S_{\mathcal{V}'}(t'_\beta) = \beta_1 \setminus getSec_{\mathcal{V}}(N_M)$. Similarly, $S_{\mathcal{V}'}(t_\alpha) = \alpha_1 \setminus getSec_{\mathcal{V}}(N_M)$, $(t_\beta \upharpoonright (Ev^{sec}_{\mathcal{V}_M(\mathcal{V})} \setminus N_\mathcal{V})) \approx_\mathcal{V} (\beta_2 \setminus N_\mathcal{V}) \approx_\mathcal{V} (t'_\beta \upharpoonright (Ev^{sec}_{\mathcal{V}_M(\mathcal{V})} \setminus N_\mathcal{V}))$ and $(t_\alpha \upharpoonright (Ev^{sec}_{\mathcal{V}_M(\mathcal{V})} \setminus N_\mathcal{V})) \approx_\mathcal{V} (\alpha_2 \setminus N_\mathcal{V})$. Since B is total in N_M and B_M includes $\approx_\mathcal{V}$ and is total in $N_\mathcal{V}$ and using the lemma, we get $(\beta_1 \cdot \alpha_1, \beta_1 \cdot S_\mathcal{V}(t'_\alpha)) \in B$, $(\beta_2 \cdot \alpha_2, \beta_2 \cdot (t'_\alpha \upharpoonright Ev^{sec}_{\mathcal{V}_M(\mathcal{V})})) \in B_M$, and $\alpha' \in S_\mathcal{V}(t'_\alpha) \parallel_S (t'_\alpha \upharpoonright Ev^{sec}_{\mathcal{V}_M(\mathcal{V})})$, as required by Definition 3.25. Moreover, $sl \preceq_C sl'$ follows directly from the definition of B'. □

Proof of Lemma 6.5. Fix $\mathcal{V} \in \mathcal{V}\mathbf{s}$, and let $G_{OD}(\mathcal{V}) = (N_{OD}, B_{OD})$. It is straightforward to prove that LTS_{OD} is secure w.r.t. $\mathcal{V}_M(\mathcal{V})$ and B_{OD}. The fact that any changes to the sequence of secret events must preserve ordered delivery is declassified in the bound. Observable communication is always preserved, due to the assumption that a *Recv* event is observable iff the corresponding *Send* event is. Moreover, LTS_{OD} accepts the neutral events of $\mathcal{V}$ at the interface: these must be *Send* events due to the assumptions of the lemma, B_{OD} is total in these events, and LTS_{OD} supports their eager insertion at any point in the trace. However, to preserve ordered delivery, it might be necessary to insert a corresponding *Recv* event later in the trace, which is neutral for the monitor. Since this only needs to happen after all observable and secret messages currently in transit between the involved activities have been delivered, LTS_{OD} silently accepts $N_\mathcal{V}$ before $Ev_{OD} \setminus (N_\mathcal{V} \cup N_{OD})$. The last condition of Definition 6.6 is trivially satisfied because $\mathcal{V}$ is invertible on Ev_{OD} by assumption. □

Proof of Theorem 6.6. In order to apply Theorem 6.3, we have to show that G_{OD} is composable. Let $\mathcal{V} \in \mathcal{V}\mathbf{s}_{OD}$ as defined in Lemma 6.5. The first condition of Definition 6.8 is satisfied due to $Ev_{OD} \cap Ev^{obs}_\mathcal{V} \subseteq Ev^{sec}_\mathcal{V}$. The assumptions on *Send* and *Recv* events in the definition of $\mathcal{V}\mathbf{s}_{OD}$ imply that all *Recv* events matching neutral *Send* events are secret and non-observable, so the second condition of Definition 6.8 is satisfied. The remaining conditions of Definition 6.8 are satisfied because $\mathcal{V}$ is invertible on Ev_{OD} and distinguishes Ev_{OD} as well as $Ev^{sec}_\mathcal{V} \cap Ev^{obs}_\mathcal{V}$, and B_{OD} is total in $N_\mathcal{V}$. Finally, the other preconditions of Theorem 6.3 follow from Lemma 6.5 and the assumptions of Theorem 6.6. □

Appendix D.

Declassification Triggers

The original formulation of BD security in [KLP14] includes an additional parameter T, a *declassification trigger*: The security criterion is only required to hold for traces t where T does not occur. Hence, as soon as the trigger occurs, the security property no longer offers any guarantees.

Definition D.1. Let $\mathcal{V}$ be a view on LTS, and $T\colon Ev \to Bool$ be a predicate on events. *LTS satisfies triggered BD security w.r.t. $\mathcal{V}$, B, and T* iff for all $t \in [\![LTS]\!]$ and $s' \in \mathsf{Sec}^*$,

$$(\mathrm{filter}(T,t) = \langle\rangle) \wedge (S_{\mathcal{V}}(t), s') \in B \longrightarrow (\exists t' \in [\![LTS]\!].\ O_{\mathcal{V}}(t') = O_{\mathcal{V}}(t) \wedge S_{\mathcal{V}}(t') = s')$$

This can be convenient to model situations where arbitrary declassification is allowed after certain events have happened. However, for simplicity, we prefer to avoid additional parameters of the model that would clutter our discussions. Instead, we use the notion of BD Security of Definition 2.2, which means we take T to be vacuously False.[1]

An interesting theoretical question is whether in general we can formulate BD security properties without the trigger, with no loss of generality. Here, we give a partially positive answer to this question, showing that the bound can be enriched to cater for any desired trigger. The transformed BD security property is *almost* equivalent to the original one, but slightly stronger (as we discuss below).

Given a BD security property as introduced above with a static trigger T, we transform it into another instance of BD security without static trigger, i.e., as given in Definition 2.2. For this purpose, we first extend the notion of *secret* by adding a Boolean value, representing whether the current event causes the trigger to fire or not. Hence, the new secrets are *pairs* of a Boolean and an original secret (or $\bot$ for events that only cause the trigger to fire, but do not contain secret information), i.e., $\mathsf{Sec}_T = Bool \times (\mathsf{Sec} \uplus \{\bot\})$. Furthermore, $isSec_T$ and $getSec_T$ are extended so that a secret with a True Boolean part is produced whenever the trigger fires.

$$Ev_T^{sec} \equiv Ev^{sec} \cup \{e \mid T(e)\}$$

$$getSec_T(e) \equiv \begin{cases} getSec(e) & \text{if } \neg T(e) \wedge e \in Ev^{sec} \\ \bot & \text{if } T(e) \end{cases}$$

[1] Except for Chapter 4, where we use triggers during parts of our discussion to keep the bounds simple.

This extended view $\mathcal{V}_T = (Ev^{obs}, getObs, Ev_T^{sec}, getSec_T)$, allows us to talk about the (non)occurrence of the trigger in the bound B_T as follows.

$$B_T \equiv \big\{(sl \restriction \mathsf{Sec}, sl' \restriction \mathsf{Sec}) \in B \mid (sl \restriction \{\bot\} = \langle\rangle) \wedge \big(sl' \restriction \{\bot\} = \langle\rangle\big)\big\}$$

By only considering secret sequences without $\bot$, we capture the fact that the notion of triggered BD Security of Definition D.1 only considers traces t where the trigger never occurs. That notion of BD Security is still implied by the transformed property:

Lemma D.1. *If LTS is BD secure w.r.t. $\mathcal{V}_T$ and B_T, then it satisfies triggered BD Security w.r.t. $\mathcal{V}$, B, and T.*

Proof. By induction on t, we have $(\text{filter}(T, t) = \langle\rangle) \longleftrightarrow (S_{\mathcal{V}_T}(t) \restriction \{\bot\} = \langle\rangle)$. Hence, for any t for which the original property requires us to find an alternative trace, we can invoke the transformed property to obtain an alternative trace t' that meets the requirements of both properties. □

The transformed property is slightly *stronger* than the original BD Security property with static trigger: The converse of the implication in the above lemma does not hold in general. This is due to the fact that triggered BD Security does not specify whether the trigger is allowed to occur in the *alternative* trace t', while the transformed bound B' specifies that it *must not* occur. The latter is necessary in the transformed setup, because otherwise, we would have to specify explicitly *when* the trigger *must* occur. This is not reasonably possible without further knowledge of the system.

These considerations suggest a strengthening of the notion of BD Security originally proposed in [KLP14] that rules out the occurrence of the trigger in the alternative trace.

Definition D.2. *LTS* satisfies *trigger-preserving BD security* w.r.t. $\mathcal{V}$, B, and T iff for all $t \in [\![LTS]\!]$ and $s' \in \mathsf{Sec}^*$,

$$\begin{aligned}&\text{filter}(T, t) = \langle\rangle \wedge S_{\mathcal{V}}(t), s') \in B \longrightarrow \\ &(\exists t' \in [\![LTS]\!].\ O_{\mathcal{V}}(t') = O_{\mathcal{V}}(t) \wedge S_{\mathcal{V}}(t') = s' \wedge \text{filter}(T, t') = \langle\rangle)\end{aligned}$$

The result of the above transformation is equivalent to this strengthened notion of BD Security:

Lemma D.2. *LTS is BD secure w.r.t. $\mathcal{V}_T$ and B_T iff it satisfies trigger-preserving BD security w.r.t. $\mathcal{V}$, B, and T.*

Proof. Analogous to Lemma D.1, but where in the reverse direction the preservation of trigger-absence guarantees that t' is also suitable for the transformed property. □

Let us discuss the difference between the original notion of BD Security of [KLP14] and its trigger-preserving variant. Both notions require that, for each modification of

the secret that is required by the bound, the system can produce an alternative trace while preserving the observation. The difference is that the original notion is slightly more flexible, in that it allows the system to let the trigger fire in the alternative trace. However, there does not seem much to be gained from this flexibility. Indeed, we have not used this flexibility in our case studies, so we could also have used the trigger-preserving variant instead. Moreover, the latter, as well as our trigger-free notion of BD Security given in Definition 2.2, has the pleasant mathematical property that it implies transitivity of the bound:

Lemma D.3. *Let B^* denote the reflexive-transitive closure of B.*

- *LTS is BD secure w.r.t. $\mathcal{V}$ and B iff it is BD secure w.r.t. $\mathcal{V}$ and B^*.*
- *LTS satisfies trigger-preserving BD Security w.r.t. $\mathcal{V}$, B, and T iff it satisfies trigger-preserving BD Security w.r.t. $\mathcal{V}$, B^*, and T.*

Proof. "$\Leftarrow$": Security w.r.t. B^* trivially implies security w.r.t. B due to $B^* \supseteq B$.

"$\Rightarrow$": We use induction on the construction of B^*. For a reflexive pair of secret sequences in B, the requirements of (trigger-preserving) BD Security are trivially satisfied by choosing the original trace t itself. In the base case of the transitive closure $(S_{\mathcal{V}}(t), sl') \in B$, we simply invoke the security property w.r.t. B. In the inductive step of the transitive closure, let $(S_{\mathcal{V}}(t), sl') \in B^*$ with $(S_{\mathcal{V}}(t), sl'') \in B$ and $(sl'', sl') \in B^*$. We apply (trigger-preserving) BD Security once to obtain a trace $t'' \in [\![LTS]\!]$ producing sl'' (and where the trigger never occurs), and then apply the induction hypothesis to obtain a $t' \in [\![LTS]\!]$ producing sl'. □

This result can be used to reduce the verification effort by focusing on a subset of the bound, and then obtaining the reflexive-transitive closure for free. This approach is used, for example, in the MAKS framework [Man00a]. Most of its Basic Security Predicates (BSPs) focus on the insertion or deletion of *one* confidential event at a time. The possibility of modifying *sequences* of confidential events follows by transitivity.

We have decided to use trigger-free BD Security in this thesis. We do not lose any expressivity by doing this, because static triggers can still be encoded using the above transformation. We gain transitivity w.r.t. the bound and simplicity of the model, because it has one parameter less than the original formulation of BD Security in [KLP14].

Bibliography

[Aci07] Onur Aciiçmez. Yet another microarchitectural attack: exploiting i-cache. In *Proceedings of the 2007 ACM workshop on Computer Security Architecture, CSAW 2007*, pages 11–18, 2007.

[ACPP11] Wihem Arsac, Luca Compagna, Giancarlo Pellegrino, and Serena Elisa Ponta. Security validation of business processes via model-checking. In *Engineering Secure Software and Systems*, number 6542 in LNCS, pages 29–42. Springer, January 2011.

[AGK+96] Gustavo Alonso, Roger Günthör, Mohan Kamath, Divyakant Agrawal, Amr El Abbadi, and C. Mohan. Exotica/fmdc: A workflow management system for mobile and disconnected clients. *Distributed and Parallel Databases*, 4(3):229–247, 1996.

[AL12] Rafael Accorsi and Andreas Lehmann. Automatic information flow analysis of business process models. In *Business Process Management - 10th International Conference, BPM 2012*, pages 172–187, 2012.

[AS87] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987.

[BFG+14] Gilles Barthe, Cédric Fournet, Benjamin Grégoire, Pierre-Yves Strub, Nikhil Swamy, and Santiago Zanella Béguelin. Probabilistic relational verification for cryptographic implementations. In *Principles of Programming Languages, POPL 2014*, pages 193–206, 2014.

[BFPR03] Annalisa Bossi, Riccardo Focardi, Carla Piazza, and Sabina Rossi. Refinement operators and information flow security. In *1st International Conference on Software Engineering and Formal Methods, SEFM 2003*, pages 44–53. IEEE Computer Society, 2003.

[BH14a] Thomas Bauereiß and Dieter Hutter. Possibilistic information flow control for workflow management systems. In *First International Workshop on Graphical Models for Security, GraMSec 2014*, pages 47–62, 2014. Extended version with proofs available at `https://www.thomas-bauereiss.de/papers/WorkflowSecurity_TR.pdf`.

[BH14b] Thomas Bauereiß and Dieter Hutter. Compatibility of safety properties and possibilistic information flow security in MAKS. In *ICT Systems*

Security and Privacy Protection - 29th IFIP TC 11 International Conference, SEC 2014, pages 250–263, 2014.

[BHLR12] Achim D. Brucker, Isabelle Hang, Gero Lückemeyer, and Raj Ruparel. SecureBPMN: Modeling and enforcing access control requirements in business processes. In *17th ACM Symposium on Access Control Models and Technologies, SACMAT 2012*, pages 123–126. ACM, 2012.

[BK85] Jan A. Bergstra and Jan Willem Klop. Algebra of communicating processes with abstraction. *Theoretical computer science*, 37:77–121, 1985.

[BMPR03] A. Bossi, D. Macedonio, C. Piazza, and S. Rossi. Secure contexts for confidential data. In *16th IEEE Computer Security Foundations Workshop, 2003*, pages 14–28, June 2003.

[BMPR05] Annalisa Bossi, Damiano Macedonio, Carla Piazza, and Sabina Rossi. Information flow in secure contexts. *Journal of Computer Security*, 13(3):391–422, 2005.

[BPPR16] Thomas Bauereiß, Armando Pesenti Gritti, Andrei Popescu, and Franco Raimondi. CoSMed: A confidentiality-verified social media platform. In *Interactive Theorem Proving - 7th International Conference, ITP 2016*, pages 87–106, 2016.

[BPPR17] Thomas Bauereiß, Armando Pesenti Gritti, Andrei Popescu, and Franco Raimondi. CoSMeDis: A distributed social media platform with formally verified confidentiality guarantees. In *IEEE Symposium on Security and Privacy, SP 2017*, pages 729–748, 2017.

[BPPR18] Thomas Bauereiß, Armando Pesenti Gritti, Andrei Popescu, and Franco Raimondi. CoSMed: A confidentiality-verified social media platform. *J. Autom. Reasoning*, 61(1-4):113–139, 2018.

[BRJ+17] Abhishek Bichhawat, Vineet Rajani, Jinank Jain, Deepak Garg, and Christian Hammer. Webpol: Fine-grained information flow policies for web browsers. In *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Proceedings, Part I*, pages 242–259, 2017.

[BRS10] Arnar Birgisson, Alejandro Russo, and Andrei Sabelfeld. Unifying facets of information integrity. In *Information Systems Security - 6th International Conference, ICISS 2010*, pages 48–65, 2010.

[BSVD09] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. PeerSoN: P2p Social Networking: Early Experiences and Insights. In *Second ACM EuroSys Workshop on Social Network Systems*, pages 46–52. ACM, 2009.

[CFK+14] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *Principles of Security and Trust - Third International Conference, POST 2014*, pages 265–284, 2014.

[CLM+09] Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng. Building secure web applications with automatic partitioning. *Commun. ACM*, 52(2):79–87, 2009.

[CM15] Stephen Chong and Ron Van Der Meyden. Using Architecture to Reason About Information Security. *ACM Trans. Inf. Syst. Secur.*, 18(2):8:1–8:30, December 2015.

[CMS09] Leucio Antonio Cutillo, Refik Molva, and Thorsten Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, page 95, 2009.

[CS10] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[dACD+14] Arthur Azevedo de Amorim, Nathan Collins, André DeHon, Delphine Demange, Catalin Hritcu, David Pichardie, Benjamin C. Pierce, Randy Pollack, and Andrew Tolmach. A verified information-flow architecture. In *Principles of Programming Languages, POPL 2014*, pages 165–178, 2014.

[DGK+13] Mads Dam, Roberto Guanciale, Narges Khakpour, Hamed Nemati, and Oliver Schwarz. Formal verification of information flow security for a simple ARM-based separation kernel. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13,*, pages 223–234, 2013.

[DP10] Dominique Devriese and Frank Piessens. Noninterference through secure multi-execution. In *IEEE Symposium on Security and Privacy, SP 2010*, pages 109–124, 2010.

[FAZ09] Philip W. L. Fong, Mohd M. Anwar, and Zhen Zhao. A privacy preservation model for Facebook-style social network systems. In *14th European Symposium on Research in Computer Security, ESORICS 2009*, pages 303–320, 2009.

[FG93] Riccardo Focardi and Roberto Gorrieri. An information flow security property for ccs. In *the Second North American Process Algebra Workshop (NAPAW'93)*, 1993.

[FG01] Riccardo Focardi and Roberto Gorrieri. Classification of Security Properties (Part I: Information Flow). In *Foundations of Security Analysis*

and Design, volume 2171 of *Lecture Notes in Computer Science*, pages 331–396. Springer Berlin / Heidelberg, 2001.

[GG16] Simon Greiner and Daniel Grahl. Non-interference with what-declassification in component-based systems. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016*, pages 253–267, 2016.

[GM82] J.A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy, SP 1982*, volume 11, 1982.

[GM84] Joseph A. Goguen and José Meseguer. Unwinding and inference control. In *IEEE Symposium on Security and Privacy, SP 1984*, pages 75–87, 1984.

[GR15] Joshua D. Guttman and Paul D. Rowe. A cut principle for information flow. In *IEEE 28th Computer Security Foundations Symposium, CSF 2015*, pages 107–121, 2015.

[HMSS07] Dieter Hutter, Heiko Mantel, Ina Schaefer, and Axel Schairer. Security of multi-agent systems: A case study on comparison shopping. *Journal of Applied Logic*, 5(2):303–332, June 2007.

[HN10] Florian Haftmann and Tobias Nipkow. Code generation via higher-order rewrite systems. In *Functional and Logic Programming, 10th International Symposium, FLOPS 2010*, pages 103–117, 2010.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HS09] Christian Hammer and Gregor Snelting. Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs. *Int. J. Inf. Sec.*, 8(6):399–422, 2009.

[HSY06] David S. Hardin, Eric W. Smith, and William D. Young. A robust machine code proof framework for highly secure applications. In *Sixth International Workshop on the ACL2 Theorem Prover and its Applications, ACL2 2006*, pages 11–20, 2006.

[Hut06] Dieter Hutter. Possibilistic information flow control in MAKS and action refinement. In *Int. Conf. on Emerging Trends in Information and Communication Security, ETRICS-2006*, volume 3995 of *LNCS*, pages 268–281. Springer, 2006.

[Jac89] J. Jacob. On the derivation of secure components. In *IEEE Symposium on Security and Privacy, SP 1989*, pages 242–247. IEEE, May 1989.

[JNM+12] Sonia Jahid, Shirin Nilizadeh, Prateek Mittal, Nikita Borisov, and Apu Kapadia. DECENT: A decentralized architecture for enforcing privacy

in online social networks. In *Tenth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2012, Workshop Proceedings*, pages 326–332. IEEE, 2012.

[JTL12] Dongseok Jang, Zachary Tatlock, and Sorin Lerner. Establishing browser security guarantees through formal shim verification. In *USENIX Security*, pages 113–128, 2012.

[KAE+10] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an operating-system kernel. *Communications of the ACM*, 53(6):107–115, June 2010.

[KAE+14] Gerwin Klein, June Andronick, Kevin Elphinstone, Toby Murray, Thomas Sewell, Rafal Kolanski, and Gernot Heiser. Comprehensive formal verification of an OS microkernel. *ACM Transactions on Computer Systems*, 32(1):2:1–2:70, February 2014.

[KHF+19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *IEEE Symposium on Security and Privacy, SP 2019*, pages 1–19, 2019.

[KLP14] Sudeep Kanav, Peter Lammich, and Andrei Popescu. A conference management system with verified document confidentiality. In *Computer Aided Verification - 26th International Conference, CAV 2014. Proceedings*, pages 167–183, 2014.

[KTB+15] Ralf Küsters, Tomasz Truderung, Bernhard Beckert, Daniel Bruns, Michael Kirsten, and Martin Mohr. A hybrid approach for proving noninterference of java programs. In *IEEE 28th Computer Security Foundations Symposium, CSF 2015*, pages 305–319, 2015.

[KWP99] Florian Kammüller, Markus Wenzel, and Lawrence C. Paulson. Locales—a sectioning concept for Isabelle. In *Theorem Proving in Higher Order Logics, TPHOLs'99*, pages 149–166, 1999.

[Lam97] Leslie Lamport. Composition: A way to make proofs harder. In *Compositionality: The Significant Difference, International Symposium, COMPOS'97*, pages 402–423, 1997.

[LBW05] Jay Ligatti, Lujo Bauer, and David Walker. Enforcing non-safety security policies with program monitors. *10th European Symposium on Research in Computer Security, ESORICS 2005*, pages 355–373, 2005.

[LGV+09] Jed Liu, Michael D. George, K. Vikram, Xin Qi, Lucas Waye, and Andrew C. Myers. Fabric: a platform for secure distributed computation and storage. In *22nd ACM Symposium on Operating Systems Principles, SOSP 2009*, pages 321–334, 2009.

[LMP12] Alexander Lux, Heiko Mantel, and Matthias Perner. Scheduler-independent declassification. In *Mathematics of Program Construction - 11th International Conference, MPC 2012*, pages 25–47, 2012.

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium, USENIX Security 2018*, pages 973–990, 2018.

[Man00a] Heiko Mantel. Possibilistic definitions of security - an assembly kit. In *13th IEEE Computer Security Foundations Workshop, CSFW '00*, pages 185–199. IEEE Computer Society, 2000.

[Man00b] Heiko Mantel. Unwinding possibilistic security properties. In *6th European Symposium on Research in Computer Security, ESORICS 2000*, volume 1895 of *LNCS*, pages 238–254. Springer, 2000.

[Man01a] Heiko Mantel. Information flow control and applications - bridging a gap. In *FME 2001: Formal Methods for Increasing Software Productivity*, pages 153–172, 2001.

[Man01b] Heiko Mantel. Preserving information flow properties under refinement. In *IEEE Symposium on Security and Privacy, SP 2001*, pages 78–91. IEEE Computer Society, 2001.

[Man02] Heiko Mantel. On the composition of secure systems. In *IEEE Symposium on Security and Privacy, SP 2002*, pages 88–101. IEEE Computer Society, 2002.

[Man04] Heiko Mantel. *A uniform framework for the formal specification and verification of information flow security.* PhD thesis, Universität des Saarlandes, 2004.

[Man05] Heiko Mantel. The framework of selective interleaving functions and the modular assembly kit. In *2005 ACM workshop on Formal methods in security engineering, FMSE 2005*, pages 53–62, 2005.

[McC87] Daryl McCullough. Specifications for multi-level security and a hook-up property. In *IEEE Symposium on Security and Privacy, SP 1987*, pages 161–166. IEEE Computer Society, 1987.

[McC90] Daryl McCullough. A hookup theorem for multilevel security. *IEEE Trans. Software Eng.*, 16(6):563–568, 1990.

[McL96] J. McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, January 1996.

[Mil80] Robin Milner. *A calculus of communicating systems.* 1980.

[MM11] Fabio Martinelli and Ilaria Matteucci. Preserving security properties under refinement. In *7th International Workshop on Software Engineering for Secure Systems, SESS 2011*, pages 15–21. ACM, 2011.

[MMB+13] Toby C. Murray, Daniel Matichuk, Matthew Brassil, Peter Gammie, Timothy Bourke, Sean Seefried, Corey Lewis, Xin Gao, and Gerwin Klein. seL4: From general purpose to a proof of information flow enforcement. In *IEEE Symposium on Security and Privacy, SP 2013*, pages 415–429, 2013.

[Mor09] Carroll Morgan. The shadow knows: Refinement and security in sequential programs. *Sci. Comput. Program.*, 74(8):629–653, 2009.

[MR07] Heiko Mantel and Alexander Reinhard. Controlling the what and where of declassification in language-based security. In Rocco De Nicola, editor, *Programming Languages and Systems*, volume 4421 of *LNCS*, pages 141–156. Springer, 2007.

[MS03] Heiko Mantel and Andrei Sabelfeld. A unifying approach to the security of distributed and multi-threaded programs. *Journal of Computer Security*, 11(4):615–676, 2003.

[MSZ06] Andrew C. Myers, Andrei Sabelfeld, and Steve Zdancewic. Enforcing robust declassification and qualified robustness. *Journal of Computer Security*, 14(2):157–196, January 2006.

[MWW+98] Peter Muth, Dirk Wodtke, Jeanine Weissenfels, Angelika Kotz Dittrich, and Gerhard Weikum. From centralized workflow specification to distributed workflow execution. *Journal of Intelligent Information Systems*, 10(2):159–184, March 1998.

[Mye99] Andrew C. Myers. Jflow: Practical mostly-static information flow control. In *Principles of Programming Languages, POPL 1999*, pages 228–241, 1999.

[NPW02] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283 of *LNCS*. Springer, 2002.

[PA17] Mathias V. Pedersen and Aslan Askarov. From trash to treasure: Timing-sensitive garbage collection. In *IEEE Symposium on Security and Privacy, SP 2017*, pages 693–709, 2017.

[PHN13] Andrei Popescu, Johannes Hölzl, and Tobias Nipkow. Formal verification of language-based concurrent noninterference. *Journal of Formalized Reasoning*, 6(1):1–30, 2013.

[PS14] Raúl Pardo and Gerardo Schneider. A formal privacy policy framework for social networks. In *Software Engineering and Formal Methods - 12th International Conference, SEFM 2014*, pages 378–392, 2014.

[RFMP07] Alfonso Rodríguez, Eduardo Fernández-Medina, and Mario Piattini. A bpmn extension for the modeling of security requirements in business processes. *IEICE Transactions*, 90-D(4):745–752, 2007.

[RS10] Alejandro Russo and Andrei Sabelfeld. Dynamic vs. static flow-sensitive security analysis. In *23rd IEEE Computer Security Foundations Symposium, CSF 2010*, pages 186–199. IEEE Computer Society, 2010.

[RS14] Willard Rafnsson and Andrei Sabelfeld. Compositional information-flow security for interactive systems. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014*, pages 277–292. IEEE, 2014.

[RS16] Willard Rafnsson and Andrei Sabelfeld. Secure multi-execution: Fine-grained, declassification-aware, and transparent. *Journal of Computer Security*, 24(1):39–90, 2016.

[San08] Thomas Santen. Preservation of probabilistic information flow under refinement. *Inf. Comput.*, 206(2-4):213–249, 2008.

[Sch00] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, February 2000.

[SJKB94] Hans Schuster, Stefan Jablonski, Thomas Kirsche, and Christoph Bussler. A client/server architecture for distributed workflow management systems. In *Third International Conference on Parallel and Distributed Information Systems (PDIS 94)*, pages 253–256. IEEE Computer Society, 1994.

[SLS06] Andreas Schaad, Volkmar Lotz, and Karsten Sohr. A model-checking approach to analysing organisational controls in a loan origination process. In *11th ACM Symposium on Access Control Models and Technologies, SACMAT 2006*, pages 139–149. ACM, 2006.

[SM03] A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[Smi09] Geoffrey Smith. On the foundations of quantitative information flow. In *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009*, pages 288–302, 2009.

[SS01] Andrei Sabelfeld and David Sands. A per model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation*, 14(1):59–91, March 2001.

[SS05] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.

[SS06] Fredrik Seehusen and Ketil Stølen. Maintaining information flow security under refinement and transformation. In *Formal Aspects in Security and Trust*, pages 143–157, 2006.

[SS09] Andrei Sabelfeld and David Sands. Declassification: Dimensions and principles. *Journal of Computer Security*, 17(5):517–548, 2009.

[Sut86] David Sutherland. A model of information. In *9th National Security Conf.*, pages 175–183, 1986.

[VIS96] Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996.

[WG08] Peter Y. H. Wong and Jeremy Gibbons. A process semantics for BPMN. In *Formal Methods and Software Engineering, 10th International Conference on Formal Engineering Methods, ICFEM 2008*, volume 5256 of *LNCS*, pages 355–374. Springer, 2008.

[Win87] Glynn Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 325–392, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[WM10] Christian Wolter and Christoph Meinel. An approach to capture authorisation requirements in business processes. *Requir. Eng.*, 15(4):359–373, 2010.

[YLGY10] Ping Yang, Shiyong Lu, Mikhail I. Gofman, and Zijiang Yang. Information flow analysis of scientific workflows. *Journal of Computer and System Sciences*, 76(6):390–402, September 2010.

[YYS12] Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. A language for automatically enforcing privacy policies. In *Principles of Programming Languages, POPL 2012*, pages 85–96, 2012.

[ZBM08] Nickolai Zeldovich, Silas Boyd-Wickizer, and David Mazières. Securing distributed systems with information flow control. In *5th USENIX Symposium on Networked Systems Design & Implementation, NSDI 2008*, pages 293–308, 2008.

[ZL97] Aris Zakinthinos and E. Stewart Lee. A general theory of security properties. In *IEEE Symposium on Security and Privacy, SP 1997*, pages 94–102. IEEE Computer Society, 1997.

[ZM03] Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *16th IEEE Computer Security Foundations Workshop (CSFW-16 2003)*, page 29. IEEE Computer Society, 2003.

[ZZNM02] Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Secure program partitioning. *ACM Trans. Comput. Syst.*, 20(3):283–328, 2002.